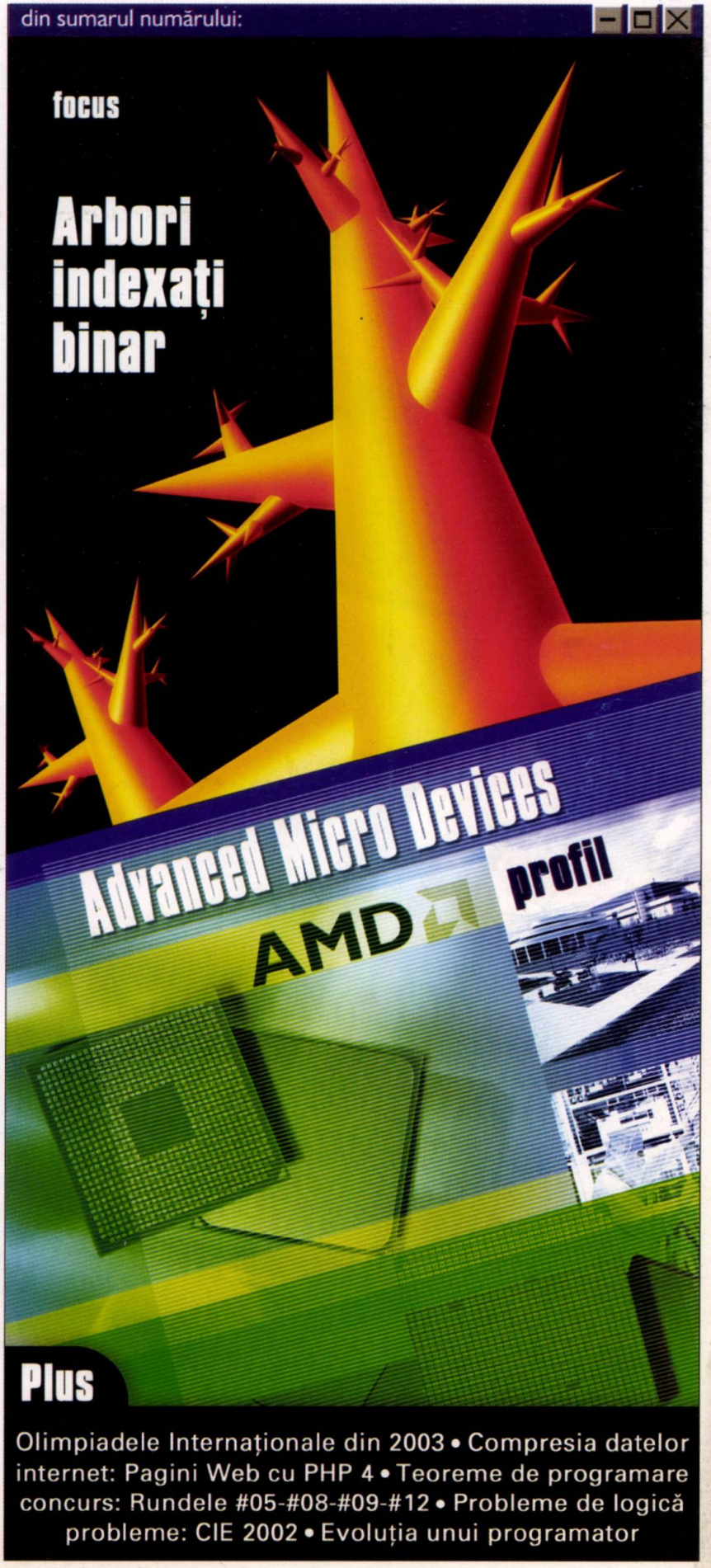
ianuarie 2003
Vol. 13/1

Lei 30.000

gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:



focus

Arbori indexați binar

Advanced Micro Devices

AMD

profil

Plus

Olimpiadele Internaționale din 2003 • Compresia datelor internet: Pagini Web cu PHP 4 • Teoreme de programare concurs: Rundele #05-#08-#09-#12 • Probleme de logică probleme: CIE 2002 • Evoluția unui programator

*a se consuma
înainte de pornirea
calculatorului

Vol. 13, Nr. 1 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. *Clara Ionescu*
Redactor-șef:
Mihai Scorțaru
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Claudiu Soroiu

Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/192.422
0265/166.516
Fax: 0265/166.290
http://www.ginfo.ro
ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 0265/162.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*", "*NetReport*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László

Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu

Agora MEDIA

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

probleme.. PAG. 5

C.I.E. 2002

~ Concurs organizat la ~
~ ASE București ~

soluții.. PAG. 6

AMINTIRI

~ soluțiile problemelor ~
~ propuse în GInfo 12/7 ~

concurs.. PAG. 9

Despre rundele #05-#08

Clasament

Enunțurile problemelor de la rundele #09-#12

babel.. PAG. 15

OLIMPIADELE internaționale din 2003

Mihai SCORȚARU

~ IOI, CEOI, BOI? ~

focus.. PAG. 16

ARBORI indexați BINAR

Mihai SCORȚARU

~ structuri avansate de date ~

internet.. PAG. 26

Pagini WEB cu PHP 4

Mihai SCORȚARU
Claudiu SOROIU

begin.. PAG. 35

TEOREME de programare

Clara IONESCU

~ teorie și practică ~

serial.. PAG. 36

COMPRESIA datelor

Claudiu SOROIU

~ primul episod ~

profil.. PAG. 42

Advanced Micro Devices

Carmen ȘUȘMAN

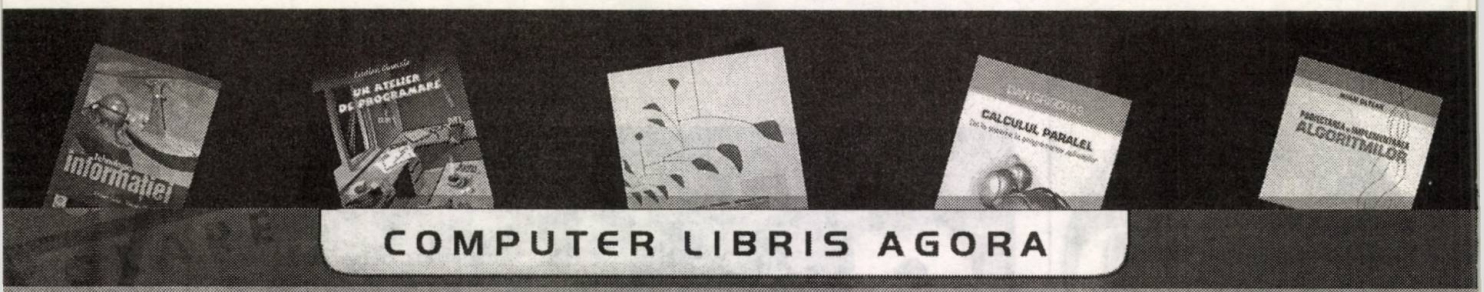
~ istoria companiei ~

enjoy.. PAG. 44

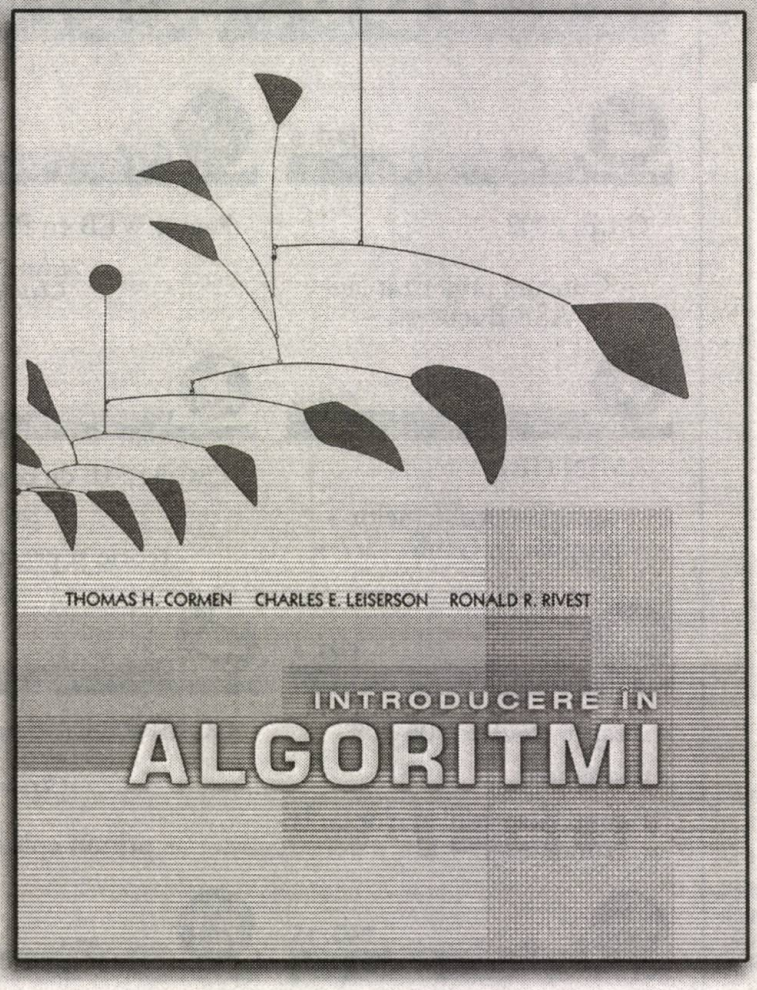
Evoluția unui PROGRAMATOR

Probleme de LOGICĂ

Mihai SCORȚARU



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne €a
analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Aparențele ÎNȘELĂ

Nu știu dacă situația este aceeași și în cazul vostru, dar mie, în prima oră de informatică din clasa a IX-a mi s-a spus că cel mai simplu "*program*" *Pascal* este:

```
Begin
End.
```

Îmi amintesc că profesoara se ferea să numească această secvență de cod "*program*" deoarece acesta "*nu face nimic*". Mai mult, el "*nu poate fi folosit la nimic*"!

Mai târziu, când am învățat să compilez programe, am scris, de curiozitate, acest program și apoi l-am executat. Într-adevăr, nu făcea nimic. Tot din curiozitate m-am uitat la fișierul executabil și am văzut că pentru a nu face nimic este nevoie de un cod executabil cu o dimensiune destul de mare. Nu îmi aduc aminte acum ce dimensiune avea atunci executabilul obținut în urma compilării cu *Turbo Pascal* 6.0, dar acum am scris din nou programul și l-am compilat cu *Borland Pascal* 7.0. Pot să vă spun că, pentru a nu face nimic, am obținut un fișier executabil de 1632 de bytes.

Pe vremea aceea mă gândeam că cei care au creat acel mediu au fost destul de neinspirați dacă au transformat cei câțiva octeți pe care i-am scris eu în peste o mie de octeți care tot nu fac nimic. Acum știu că problema nu se pune așa, dar impresia rămâne aceeași.

Acum, după aproape zece ani de când am văzut pentru prima dată acest program, mi-am dat seama că poate fi folosit la ceva și chiar "*își face foarte bine treaba*".

Evident, nici acum programul propriu-zis nu face nimic. ***Totuși, se poate face ceva cu acest program!***

Unii dintre voi știu că, pentru a corecta soluțiile trimise de participanții la concursul de programare "*Bursele Agora*" folosim un program de evaluare creat de noi. Unul dintre aspectele importante care trebuie luate în considerare în momentul creării unui astfel de produs *software* este măsurarea timpilor de execuție ai programelor concurenților.

Pentru a măsura acest timp "*neam uitat la ceas*" înainte să lansăm în execuție programul încă o dată și după ce execuția acestuia s-a terminat. Pare normal să considerăm că timpul de execuție este dat de intervalul de timp scurs între cele două momente la care "*ne-am uitat la ceas*". Totuși, ne-am dat seama că nu e chiar așa. Nu luăm în considerare timpul de care are nevoie sistemul de operare pentru a lansa în execuție programul și pentru a elibera resursele alocate programului. Este vorba de doar câteva sutimi de secundă, dar și acestea contează.

Următoarea abordare a fost să folosim informațiile furnizate de sistemul de operare *Windows* referitoare la timpii de execuție ai proceselor create pentru a rula programele. Din nefericire, s-a dovedit că *Windows*-ul nu este "*de încredere*", iar folosirea primei variante ducea la o precizie mai mare.

A treia abordare (pe care am încercat-o de curând) constă în încercarea de a determina timpul necesar sistemului de operare pentru lansarea în execuție a programului și eliberarea resurselor alocate. Cred că ați ghicit! Se folosește programul "*care nu face nimic*" și se măsoară timpul de execuție al acestuia. Datorită faptului că, practic, programul nu execută nici o operație, nu se consumă timp pentru nimic altceva. Așadar, din timpul de execuție măsurat pentru programul unui concurent scădem timpul de execuție măsurat pentru programul care nu face nimic și obținem timpul real (cu o acuratețe foarte mare) de execuție a programului concurentului.

Iată cum acel program de care am auzit în prima oră de informatică se dovedește a fi util într-o aplicație destul de complicată. Evident, există și alte posibilități de determinare exactă a timpului de execuție, dar aceasta surprinde prin simplitatea și ineditul ei.

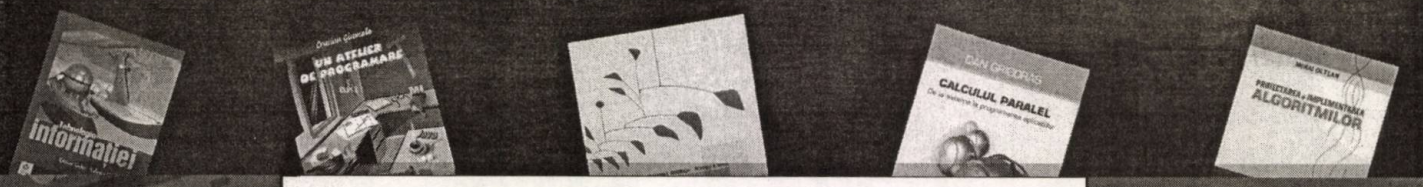
Nu mă mai mir acum că o mulțime de rezultate matematice care par pur teoretice în momentul descoperirii lor își găsesc utilitate practică după zeci sau sute de ani. Nu mă mai mir că la școală se învață tot felul de algoritmi care mai de care mai ciudați despre care se crede că nu folosesc la nimic.

Se spune că studiul algoritmilor de la școală este inutil fiindcă nu vor fi folosiți după ce elevul termină școala. Și, oricum, dacă chiar are nevoie de ei îi poate găsi în cărți. Personal, nu am văzut nici o carte în care să apară programul care nu face nimic. Chiar dacă există astfel de cărți sunt convins că în ele scrie și că "*nu e bun de nimic*".

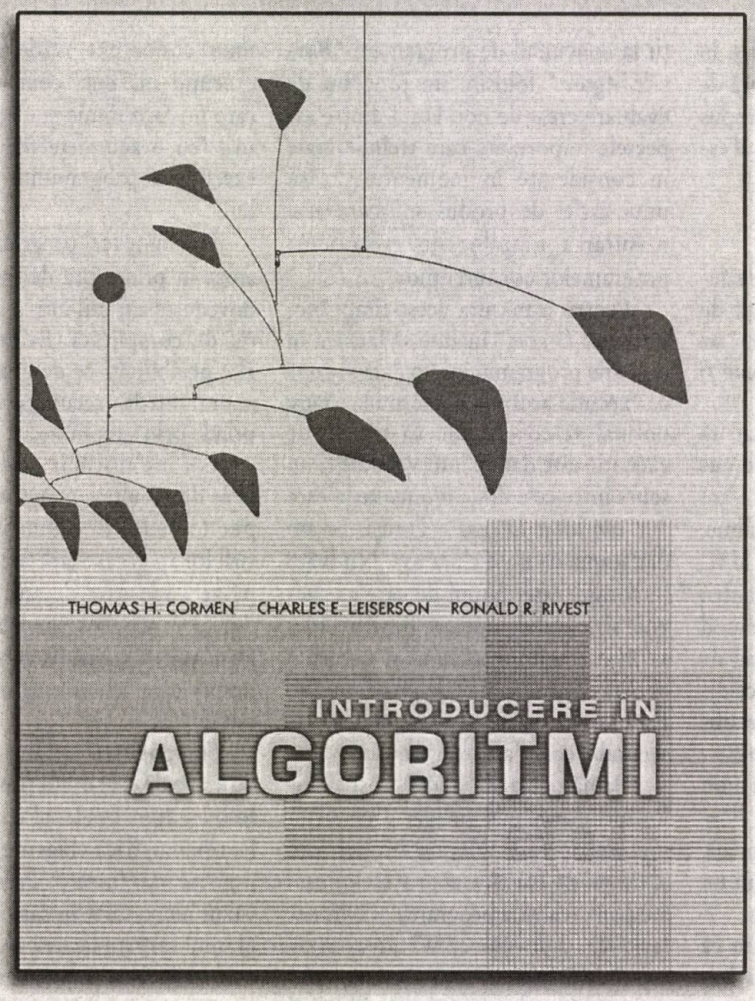
Așadar, chiar și cele mai banale programe se pot dovedi utile. Cu siguranță, unele dintre cele mai complicate vă vor fi utile în viitor. Cred că nu este cazul să ignorăm algoritmii. Limbajele de programare apar și dispar, dar algoritmii rămân cam aceeași.

În concluzie, nu cred că se poate spune despre nici un program că este inutil. Este posibil să îi găsim o utilitate atunci când ne așteptăm mai puțin. Aparențele înșeală!

Mihai Scorțaru



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

C.I.E. 2002

Vă prezentăm în continuare enunțurile celor trei probleme propuse spre rezolvare la ediția din anul 2002 a concursului de programare organizat de Academia de Studii Economice din București.

P010301: Partiție

Șeful de catedră a primit o donație de n calculatoare noi, pe care vrea să le împartă celor mai apropiați prieteni din catedră. În acest scop el a notat pe o hârtie numele celor k prieteni în ordinea importanței lor pentru el și ar vrea ca:

- fiecare prieten să primească cel puțin un calculator;
- un prieten nu poate primi mai multe calculatoare decât un altul care are numărul de ordine mai mic.

Ajutați-l pe șef, scriind un program care calculează numărul posibilităților de a împărți cele n calculatoare celor k prieteni pe care îi are în catedră, respectând cele două restricții stabilite.

Date de intrare

Fișierul de intrare **BOSS.IN** conține valorile n și k, separate printr-un singur spațiu.

Date de ieșire

La ieșirea standard va fi afișat numărul posibilităților pe care le are șeful de a împărți calculatoarele.

Restricții

- $1 \le n \le 100$;
- $1 \le k \le 10$.

Exemplu

BOSS.IN	*Ecran*
10 3	8

P010302: Procese și procesoare

Pentru executarea a N procese avem la dispoziție K procesoare. Durata de executare a fiecărui proces este de o unitate de timp. Fiecare procesor poate executa la fiecare moment de timp doar un singur proces și îl execută în întregime. Executarea unui proces poate fi condiționată de executarea în prealabil a unor alte procese. Dându-se legăturile logice între procese, se cere timpul minim în care ansamblul celor K procesoare poate executa cele N procese.

Date de intrare

Numele fișierului de intrare va fi citit de la intrarea standard. El conține pe prima linie valorile N și K, iar pe următoarele linii câte o pereche de numere I și J cu semnificația că procesul I trebuie executat înaintea procesului J; $I, J \in \{1, 2, ..., N\}$. Numerele de pe fiecare linie sunt despărțite între ele printr-un spațiu. Sfârșitul fișierului de intrare este constituit de prima linie în care apare numărul 0.

Date de ieșire

Programul trebuie să tipărească la ieșirea standard un singur număr natural care reprezintă timpul minim necesar executării tuturor proceselor. Dacă problema nu are soluție, atunci va fi tipărită valoarea 0.

Exemplu

Intrare	**PROCES.IN**	*Ecran*
PROCES.IN	6 3 3 2 2 1 4 2 6 5 0 0	3

P010303: Matrice binară

Se consideră o matrice A cu m linii și n coloane, cu elemente din mulțimea {0, 1}. Matricea nu conține linii identice sau linii care conțin mai puțin de două elemente egale cu 0. Se cere să se determine, dacă este posibil, o reordonare a coloanelor care să aducă elementele egale cu 1 de pe fiecare linie pe poziții consecutive.

Date de intrare

Numele fișierului de intrare va fi citit de la intrarea standard. Pe prima linie a fișierului apar valorile m și n, iar pe fiecare dintre următoarele m linii apare câte o linie a matricei A; elementele fiecărei linii sunt separate prin spații.

Date de ieșire

Fișierul de ieșire se numește **MATBIN.OUT**. Prima sa linie va conține fie numărul 1, fie numărul 0, după cum problema are sau nu soluție. Dacă problema are soluție, pe linia următoare va fi scrisă ordinea în care trebuie considerate coloanele astfel încât în noua matrice pe fiecare linie elementele egale cu 1 să apară pe poziții consecutive.

Exemplu

Intrare	**MATBIN.IN**	**MATBIN.OUT**
MATBIN.IN	3 6 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1	1 5 1 6 2 4 3

AMINTIRI

Prezentăm acum câteva detalii referitoare la modul în care pot fi rezolvate problemele propuse în numărul 12/7 (noiembrie 2002) al Gazetei de Informatică.

P070201: Influențe

O modalitate simplă de verificare a existenței a două persoane care se pot influența reciproc (fără a folosi cunoștințe avansate de programare) se bazează pe construirea unei liste (pentru fiecare persoană) care conține numerele de ordine ale persoanelor influențate direct sau indirect de o anumită persoană.

Putem spune că va exista o influență reciprocă în momentul în care o persoană se influențează indirect pe ea însăși (numărul ei de ordine apare în propria listă).

Pentru a construi mai ușor aceste liste le vom păstra sub forma unor mulțimi deoarece o persoană poate apărea o singură dată într-o astfel de listă.

Pentru construirea listelor vom avea în vedere faptul că o persoană *i* care influențează o persoană *j* va influența indirect toate persoanele influențate de *j*.

Vom parcurge lista perechilor (i, j) cu semnificația "*persoana i este influențată de persoana j*" și pentru fiecare astfel de pereche vom adăuga în lista (mulțimea) corespunzătoare persoanei *j* toate persoanele din lista (mulțimea) corespunzătoare persoanei *i*. Se observă că această operație constă într-o simplă reuniune de mulțimi.

După parcurgerea listei vom repeta acest pas până în momentul în care nu vom mai efectua nici o modificare a mulțimilor (nici o reuniune nu duce la mărirea numărului de elemente a vreunei mulțimi).

În acest moment vom cunoaște numerele de ordine ale tuturor persoanelor influențate de fiecare persoană *i*.

Dacă în mulțimea corespunzătoare unei persoane *i* apare și numărul *i*, atunci cu siguranță există o influență reciprocă și algoritmul se încheie.

Dacă nu a fost detectată nici o influență reciprocă, atunci cea mai influentă persoană este cea căreia îi corespunde mulțimea cu cel mai mare număr de elemente.

În continuare vom descrie modul în care poate fi determinat cel mai lung lanț al slăbiciunilor. Pentru început trebuie să observăm faptul că este posibil ca acest lanț să nu înceapă cu cea mai influentă persoană.

Pentru fiecare persoană vom construi cel mai lung lanț care începe cu acea persoană și apoi vom alege cel mai lung lanț dintre lanțurile construite.

Inițial, vom ști doar că lanțurile corespunzătoare persoanelor care nu influențează nici o altă persoană au lungimea 1. Știm cu siguranță că există astfel de persoane deoarece nu există influențe reciproce.

În continuare, la fiecare pas, dacă observăm că o persoană *i* influențează o persoană *j* al cărei lanț a fost determinat, atunci putem adăuga persoana *i* în fața acestui lanț, lungimea lui crescând cu o unitate. Evident, s-ar putea ca pentru persoana *i* să existe deja un lanț mai lung sau egal. În această situație, noul lanț este ignorat.

Vom parcurge de mai multe ori lista influențelor, până în momentul în care o parcurgere care nu duce la determinarea nici unui lanț care nu este ignorat.

După terminarea parcurgerilor vom alege cel mai lung lanț și îl vom afișa.

Observație

Există multe alte soluții mai performante, dar care necesită cunoștințe de algoritmică ce depășesc nivelul clasei a IX-a.

De exemplu, se poate construi un graf al influențelor; graful este orientat, fiecare nod reprezintă o persoană, iar pentru fiecare pereche vom avea câte un arc. Arcele vor fi de la nodul corespunzător persoanei care influențează la nodul corespunzător unei persoane pe care o influențează direct.

Vor exista influențe reciproce dacă graful este ciclic, cea mai influentă persoană este cea corespunzătoare nodului din care se poate ajunge în cele mai multe noduri urmând arcele grafului, iar cel mai lung lanț al slăbiciunilor este dat de cel mai lung traseu dintre două noduri ale grafului.

P070202: Păsărească

Dacă trebuie să traducem din română în păsărească, atunci vom parcurge cuvintele caracter cu caracter. În cazul în care caracterul curent nu este o vocală, atunci acesta este scris în fișierul de ieșire. În caz contrar (caracterul curent este o vocală *v*), în fișierul de ieșire vor fi scrise trei caractere (vocala *v*, caracterul `'p'` și din nou vocala *v*).

Dacă trebuie să traducem din păsărească în română, vom parcurge și în acest caz cuvintele caracter cu caracter.

În cazul în care caracterul curent nu este o vocală, atunci acesta este scris în fișierul de ieșire. În cazul în care caracterul curent este o vocală *v*, atunci aceasta este scrisă în fișierul de ieșire, iar următoarele două caractere sunt ignorate (acestea sunt, obligatoriu, litera `'p'` și din nou vocala *v*).

P070203: Piramida

Vom determina elementele șirului până în momentul în care vom ajunge sau vom trece de poziția *q*.

Pentru aceasta vom păstra întotdeauna poziția în șir la care începe ultima linie a piramidei și vom determina următoarele elemente (pe baza regulii din enunț). Vom repeta procedeul până în momentul în care este determinat elementul de pe poziția *q*.

În continuare vom considera numărul în baza 4 format din cifrele de pe pozițiile cuprinse între *p* și *q* și îl vom converti în baza 10.

P070204: Numere

Pentru a verifica dacă un număr are aspect de munte vom parcurge cifrele sale atât timp cât ele se află în ordine crescătoare. Dacă următoarea cifră este mai mică decât cifra curentă, atunci vom continua parcurgerea cifrelor atât timp cât ele se află în ordine descrescătoare. Dacă ajungem la ultima cifră și aceasta respectă condiția cerută (este mai mică decât predecesoarea sa), atunci numărul are aspect de munte.

Verificarea faptului că un număr are aspect de vale se realizează similar, singura diferență fiind că cifrele cu care începe numărul trebuie să fie în ordine descrescătoare, iar celelalte în ordine crescătoare.

Evident, la începutul numărului trebuie să fie cel puțin două cifre în ordine crescătoare/descrescătoare, iar la sfârșit cel puțin două în ordine descrescătoare/crescătoare. Vom construi un șir care conține numerele cu aspect de munte din intervalul $[a, b]$ și unul care conține numerele cu aspect de vale din acest interval.

În continuare vom considera toate perechile de numere din același șir și vom verifica dacă perechea curentă respectă condiția din enunț (oglinditul sumei numerelor este suma oglinditelor lor).

În momentul în care detectăm prima astfel de pereche, numerele vor fi scrise în fișierul de ieșire și vom opri execuția programului.

P070205: Cel mai mare divizor comun

Pentru a determina cel mai mare divizor comun a două numere vom folosi algoritmul lui *Euclid*.

Vom determina inițial cel mai mare divizor comun d_2 al primelor două numere din șir (a_1 și a_2), apoi vom determina cel mai mare divizor comun d_3 al numerelor d_2 și a_3. Vom continua determinarea celor mai mari divizori comuni d_i ai numerelor d_{i-1} și a_i.

Se observă că d_i este cel mai mare divizor comun al primelor i numere din șir.

Cel mai mare divizor comun al celor n numere din șir va fi d_n.

Pentru a determina divizorii primi ai acestui număr vom aplica următorul algoritm:

```
i ← 2
cât timp i ≤ d_n execută
  dacă rest[d_n / i] = 0 atunci
    scrie i
    cât timp rest[d_n / i] = 0 execută
      d_n ← d_n / i
    sfârșit cât timp
  sfârșit dacă
  i ← i + 1
sfârșit cât timp
```

Algoritmul anterior determină cel mai mic divizor i prim al valorii d_n și apoi împarte valoarea d_n la i până în momentul în care i nu mai este divizor al lui d_n. Astfel, noua valoare d_n nu va avea nici un divizor mai mic decât i.

În continuare se va determina următorul divizor (mai mare decât i) al noii valori. Acesta va fi cu siguranță prim deoarece dacă ar fi de forma $p \cdot q$ atunci p și q ar fi fost determinați ca divizori anterior și valoarea d_n ar fi fost împărțită până în momentul în care nu i-ar mai fi avut ca divizori.

Se continuă același procedeu până în momentul în care valoarea divizorului testat depășește valoarea curentă d_n.

Evident, există și posibilitatea de a testa toate numerele cuprinse între 2 și valoarea inițială d_n, dar algoritmul este mult mai lent. Pentru fiecare divizor detectat al lui d_n se verifică dacă acesta este prim.

Acest algoritm poate fi îmbunătățit dacă se consideră doar numerele din intervalul cuprins între 1 și rădăcina pătrată a valorii d_n.

În acest caz, pentru fiecare divizor i din interval se va testa atât primalitatea sa, cât și primalitatea valorii d_n / i.

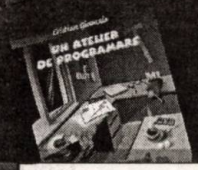
Algoritmul va funcționa corect chiar dacă d_n este un număr prim deoarece d_n va fi detectat ca divizor chiar la început (la primul pas) când se va testa primalitatea numărului 1 și cea a numărului $d_n / 1 = d_n$.

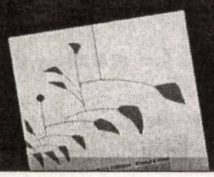
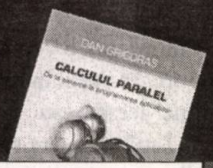
P070206: Numere deosebite

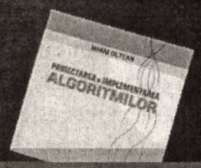
Pentru a verifica dacă un număr n este deosebit vom parcurge numerele n - 1, n - 2, etc. Pentru fiecare astfel de număr vom verifica dacă prin operația de adunare dintre număr și suma cifrelor sale se obține exact valoarea n.

Dacă valoarea n nu se obține când ajungem la numărul 0, atunci numărul n nu este deosebit.

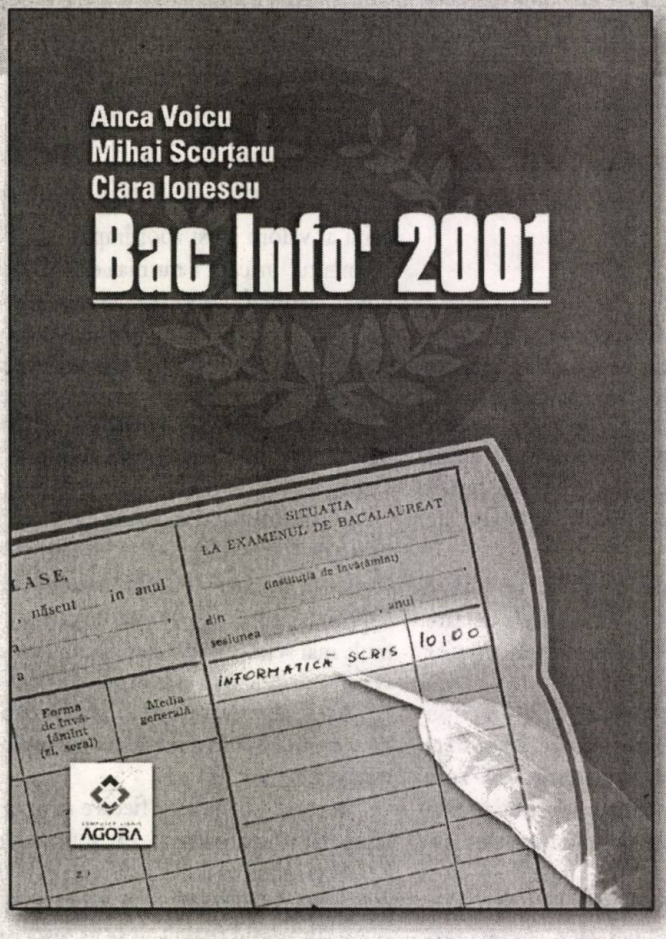
Datorită faptului că valoarea n este cuprinsă între 1 și 1000, suma cifrelor numerelor considerate este cel mult 26 (pentru $m = 999$ avem $m + S(m) = 999 + 27 = 1026 > 1000$; pentru $m = 899$ avem $m + S(m) = 899 + 26 = 925 < 1000$). Așadar putem lua în considerare numerele cuprinse între $\max(n - 26, 0)$ și n. Evident, nu are rost să testăm și numere negative.





COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Rundele #05 - #08

Vom prezenta în cadrul acestui articol câteva informații referitoare la desfășurarea a patru runde ale concursului de programare organizat de revista noastră. De asemenea, veți putea vedea clasamentul primilor o sută clasați după desfășurarea primelor opt runde.

P080215: Perechi

Algoritmul de rezolvare a acestei probleme este foarte simplu: se pornește cu cel mai mare număr ($2 \cdot n$) și se caută perechea lui, parcurgând valorile în ordine descrescătoare. După determinarea numărului pereche, acesta se marchează ca fiind utilizat.

Se continuă cu următoarele valori neutilizate ($2 \cdot n$ -1, $2 \cdot n$ - 2, ...) pentru care se determină perechile în același mod. Numerele pereche determinate la fiecare pas se marchează ca fiind utilizate și nu mai sunt utilizate pentru determinarea unei alte perechi.

În final, se vor obține toate cele n perechi. Deși nu există o demonstrație matematică a acestui fapt, algoritmul descris determină toate perechile pentru toate valorile n cuprinse între 1 și 16383 (redacția *GInfo* a testat algoritmul pentru toate aceste valori).

Tot din testele efectuate în redacție a reieșit faptul că, parcurgând valorile în ordine crescătoare (de la 1 la $2 \cdot n$), nu sunt determinate toate cele n perechi.

Problema a fost rezolvată corect de către 30 dintre participanții la această ediție a concursului organizat de *GInfo*. Aceștia sunt: *Adrian Cârcu (Bistrița), Alexandru Tomescu (București), Andrei Matei (București), Aurelian Marinescu (Brașov), Bogdan Nicolae (București), Constantin Jucovschi (Chișinău, Republica Moldova), Cornel Mihăilă (Brașov), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Daniel Nechifor (Vaslui), Dragoș Scripniciuc (Timișoara), Dumitru Codreanu (Chișinău), Florin Antonescu (Băicoi), Florin-Cezar Belghiru (Alexandria), Ioan Stan (Aiud), Ionuț-Bogdan Petre (Plopeni), Iulian Blajut (Bacău), Ivan Kazmenko (Sankt Petersburg, Federația Rusă), Lucian Suditu (Vaslui), Mihail Popa (București), Mihail-Cosmin Pițt-Rada (Drobeta Turnu-Severin), Mircea Digulescu (București), Mugurel-Ionuț Andreica (București), Radu Berinde (București), Radu Ștefan (Brașov), Radu Vatavu (Suceava), Răzvan Albu (București), Sorin-Ionuț Balasioiu (Craiova), Ștefan Ciobâcă (Suceava) și Ștefăniță Fechete (Suceava).*

Pentru această problemă, 89 dintre concurenții înscriși au trimis soluții care au fost acceptate.

P070216: Numere prime în 3D

Această problemă este o generalizare a unei probleme propuse spre rezolvare la o ediție anterioară a concursului nostru. Dacă una dintre dimensiuni este 1, problema se reduce la cea propusă anterior, singura diferență fiind faptul că sunt folosite numere prime în loc de numere impare.

Mulți concurenți au rezolvat problema tridimensională folosind ca valori ale tabloului numerele prime 7, 11 și 17. Singurul care a reușit să determine o configurație în care care se folosește și valoarea 23 a fost ***Ivan Kazmenko*** din *Rusia*, singurul concurent care a obținut punctajul maxim.

Au fost și câteva teste care corespundeau problemei bidimensionale. Aceasta a fost rezolvată folosind valorile 3, 5 și (foarte rar) 7.

Lista primilor 20 de concurenți clasați în "concursul de euristici" de la această rundă este prezentată în continuare:

1. Ivan Kazmenko, Federația Rusă	***100***
2. Ciprian Cană, Vaslui	***95***
3. Mugurel-Ionuț Andreica, București	***94***
3. Radu Ștefan, Brașov	***94***
3. Marian-Gheorghe Raica, Drăgușești	***94***
6. Csaba Pătcaș, Cluj-Napoca	***91***
6. Dumitru Codreanu, Republica Moldova	***91***
6. Mircea Digulescu, București	***91***
6. Silvestru-Cosmin Negruțeri, Bistrița	***91***
6. Slotta Johannes, Germania	***91***
11. Adrian Cârcu, Bistrița	***90***
11. Andrei Homescu, Târgu Jiu	***90***
11. Andrei Matei, București	***90***
11. Florin Antonescu, Băicoi	***90***
11. Ionuț-Bogdan Petre, Plopeni	***90***
11. Iuliu Konya, Baia Mare	***90***
17. Ekaterina Pavlova, Federația Rusă	***88***
18. Daniel Nechifor, Vaslui	***87***
19. Bogdan Nicolae, București	***82***
19. Radu Vatavu, Suceava	***82***

La această rundă au fost evaluate soluțiile trimise de 46 dintre concurenții înscriși la această ediție a concursului.

P070217: Factorizare

Pentru această problemă puteau fi folosite diferite metode de factorizare, majoritatea dintre ele neimplicând scrierea unui program propriu, ci folosirea celor disponibile pe *Web*.

Primele patru teste au fost foarte simple, factorizarea numerelor din fișierele de intrare fiind foarte simplă. Pentru cel de-al doilea test numărul era prim.

Următoarele patru teste cer factorizarea unor numere care ar fi trebuit să fie cunoscute: elemente ale șirului lui *Fibonacci*. Factorizările primelor câteva sute de termeni ale acestui șir sunt disponibile *online*, deci printr-o simplă căutare puteau fi determinate soluțiile. Numărul din testul #06 era prim.

Pentru testele #09 - #12 am folosit alte valori celebre și anume numerele *Mersenne* (cele de forma 2^n - 1). Pentru primele trei dintre aceste teste am ales trei numere *Mersenne* și le-am înmulțit în cele trei moduri posibile. Numărul din testul #12 era un număr *Mersenne* prim. Trebuie menționat faptul că funcția `IsProbablePrime()` din *Java* "susține" că acest număr este compus.

Următoarele opt teste au fost construite alegând numere prime aleatoare și înmulțindu-le. Doar ultimul test a ridicat probleme deosebite, cei doi factori având fiecare câte 50 de cifre.

Până la urmă, doar doi concurenți (***Andrei Matei*** și ***Mircea Digulescu***, ambii din *București*) au reușit să trimită soluții corecte pentru toate cele douăzeci de teste. Un număr destul de mare de concurenți (16) au reușit să găsească soluțiile pentru 19 dintre aceste teste. Aceștia sunt *Adrian Cârcu (Bistrița), Alexandru Sofronia (Bacău), Alexandru Stan (Ploiești), Alexandru Tomescu (București), Bogdan Nicolae (București), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Ionuț-Bogdan Petre (Plopeni), Iuliu Konya (Baia Mare), Maximilian Machedon (București), Mihail Popa (București), Mircea Pasoi (Ploiești), Mohamed Taha (Alexandria, Egipt), Mugurel-Ionuț Andreica (București), Radu Vatavu (Suceava)* și *Silvestru-Cosmin Negrușeri (Bistrița)*.

Numărul concurenților care au trimis fișiere de ieșire pentru această problemă a fost: 72.

P070218: Graf

Se observă că dacă se cunoaște costul unei muchii dintr-o componentă conexă a grafului, costurile celorlalte muchii pot fi determinate prin simple împărțiri. Așadar, numărul de apeluri ale funcției `CheckEdgeWeight()` este egal cu numărul componentelor conexe din graf. Doar pentru primele patru teste au fost permise mai multe apeluri ale acestei funcții.

Datorită faptului că fiecare nod are costul egal cu produsul a două numere prime, costurile celor două muchii adiacente se determină folosind o factorizare. Se alege apoi unul dintre cei doi factori și se verifică dacă este costul uneia dintre cele două muchii. Dacă răspunsul este negativ, știm cu siguranță că muchia are costul egal cu celălalt factor. Astfel, printr-un apel al funcției `CheckEdgeWidth` am determinat costul unei muchii și, foarte simplu, costul tuturor celorlalte muchii din aceeași componentă conexă.

Limita de timp admisă pentru execuția programului a fost foarte strictă, motiv pentru care erau necesare unele optimizări.

În primul rând, pentru construirea grafului, nu mai trebuia verificată existența unei muchii care are ca extremitate un nod care este deja extremitate pentru două muchii determinate anterior.

În al doilea rând, factorizarea trebuia să fie cât mai rapidă. Pentru aceasta trebuie verificată divizibilitatea costului unui nod doar cu numerele prime mai mici sau egale cu rădăcina pătrată a acestui cost. Așadar, va trebui folosit un șir al numerelor prime. Acest șir putea fi construit la începutul programului sau inclus direct în sursă sub forma unui vector de constante.

Problema s-a dovedit a fi relativ dificilă (mai ales datorită optimizărilor necesare), doar nouă dintre cele 53 de rezolvări trimise fiind punctate cu 100 de puncte. Concurenții care au primit punctajul maxim sunt:

Csaba Pătcaș, Cluj-Napoca
Daniel Nechifor, Vaslui
Dan-Ionuț Fechete, Suceava
Ivan Kazmenko, Sankt Petersburg
Maximilian Machedon, București
Mircea Digulescu, București
Mugurel-Ionuț Andreica, București
Ralph Bottesch, Sibiu
Ștefan Ciobâcă, Suceava

Despre concurs

Mai mulți concurenți au fost descalificați la runda #07 datorită faptului că au trimis fișiere de ieșire cu greșeli identice. În unul dintre cazuri era greșită chiar prima linie a fișierului de ieșire, la același test.

Am avut mai puține probleme datorate arhivelor netransferate corect. Din nefericire, a apărut o altă problemă, la runda #08. Am primit mai multe arhive care conțineau, în locul sursei, fie *unit*-ul demonstrativ, fie un program cu ajutorul căruia puteau fi generate fișiere de inițializare, fie un executabil (care nu funcționa corect datorită faptului că, în timpul evaluării, biblioteca folosea în momentul inițializării fișierul **`GRAPH.IN`** și nu **`GRAPH.TXT`**, cum era cazul bibliotecii demonstrative).

Vă rugăm, din nou, să vă asigurați (atât cât este posibil) că arhiva pe care o trimiteți este corectă.

În cazul în care vom primi o arhivă incorectă până în ziua de vineri, la ora 12:00, vom trimite mesaje de avertizare și vom accepta o a doua arhivă care, sperăm, va fi corectă.

Pe pagina alăturată este prezentată lista primilor 100 clasați (de fapt 104) după primele opt etape ale concursului de programare ***Bursele Agora***.

Mulțumim tuturor concurenților pentru participarea la concursul nostru!!!

Clasament după 8 runde

1. Adrian Cârcu, Bistrița *774*
2. Mircea Digulescu, București *738*
3. Ivan Kazmenko, Sankt Petersburg - Federația Rusă *721*
4. Mugurel-Ionuț Andreica, București *704*
5. Aurelian Marinescu, Brașov *656*
6. Andrei Matei, București *652*
7. Dumitru Codreanu, Chișinău - Republica Moldova *638*
8. Radu Vatavu, Suceava *618*
9. Dan Ghinea, București *594*
10. Bogdan Nicolae, București *590*

11. Cosmin Negruşeri, Bistrița 587
12. Csaba Pătcaș, Cluj-Napoca 584
13. Ciprian Cană, Vaslui 541
14. Daniel Nechifor, Vaslui 540
15. Ștefan Ciobâcă, Suceava 535
16. Iulian Blajut, Bacău 515
17. Cornel Mihăilă, Brașov 495
18. Ionuț-Bogdan Petre, Plopeni 483
19. Andrei Homescu, Târgu Jiu 467
20. Lucian Suditu, Vaslui 430
21. Dan-Ionuț Fechete, Suceava 420
22. C-tin Jucovschi, R. Moldova 397
23. Tiberiu Chiorean, Baia Mare 394
24. Victor Costan, București 390
25. Marian Raica, Drăgușești 367
26. Iuliu Konya, Baia Mare 365
27. Florin Antonescu, Băicoi 339
28. Răzvan Albu, București 337
29. Ștefăniță Fechete, Suceava 333
29. Mihail Popa, București 333
31. C. Bojinovici, București 331
32. Radu Berinde, București 324
33. Marius Nicolae, Slatina 316
34. M. Machedon, București 313
35. Andrei Ion, Constanța 311
36. Radu Ștefan, Brașov 294
36. Silviu Hagea, Călărași 294
38. Alin Coman, Târgu Jiu 285
39. Alexandru Sofronia, Bacău 268
40. M. Piț-Rada, Dr. Tr. Severin 264
41. Alex. Tomescu, București 260
42. Marius Moraru, Buftea 256
43. Bogdan Pănuș, Satu Mare 248
44. Alexandru Moșoi, Bacău 246
44. Csaba András, Oradea 246
46. Marius Airinei, Bacău 244
47. Mircea Pasoi, Ploiești 236
48. Angelo Lupuț, Satu Mare 235
49. Vlad Dascălu, Bacău 226
50. Călin Bindea, Bistrița 216
51. Călin Ciutu, București 214
52. Sorin Balasioiu, Craiova 200
53. Slotta Johannes, Germania 197
54. Alex. Constantin, Pitești 196
55. Florin Belghiru, Alexandria 182
56. Alex. Constantin, Ploiești 164
57. Mohamed Taha, Egipt 156
58. Saratov Team, Rusia 155
58. Ștefan Bucur, București 155
60. Natalia Baranova, Rusia 154
61. Valentin Bran, Satu Mare 151
62. Ralph Bottesch, Sibiu 146
63. Bogdan Chiriţoiu, Ploiești 143
64. Daniel Păsăilă, Suceava 141
64. Constantin Răuțu, Botoșani 141
66. Vincent Groenhuis, Olanda 137
67. Adrian Nasui, Slobozia 133
68. Ioan Stan, Aiud 124
69. Miron Emilian, Bacău 120
70. Lego Haryanto, SUA 119
71. Ștefan Nour, Piatra Neamț 114
71. D. Scripniciuc, Timișoara 114
73. Andrei Gheorghe, București 111
74. Ciprian Baicu, București 109
74. Alexandru Ivan, București 109
76. Alpár Perini, Sf. Gheorghe 108
77. Ágota Turánicz, Cluj-Napoca 107
78. Mircea-Șerban Savin, Iași 106
78. Răzvan Tigaeru, Călărași 106
80. Cosmin Raianu, Constanța 104
80. Ionuț Ene, Turturești 104
80. Victor Vernescu, Constanța 104
83. Laura Drăgoi, Prahova 101
84. Adrian Balașko, Zalău 100
84. Adrian Dumitran, București 100
84. Al. Tuztzi Sande, Lindiceni 100
84. Alexandru Radu, București 100
84. Andrei Benea, Focșani 100
84. Andrei Csibi, Cluj-Napoca 100
84. Andrei Negoiță, Bacău 100
84. Ciprian Pop, Baia Mare 100
84. C. Alexandrescu, București 100
84. Erdal-Medayin Rasid, Tulcea 100
84. Eszter Tasi, Sf. Gheorghe 100
84. G. Pawel, Polonia 100
84. Ioan Gruia, București 100
84. Lucian Vancea, Blaj 100
84. Marius Nuca, Măgura 100
84. Daniel Dumitran, București 100
84. Răzvan Dinu, Ploiești 100
84. Sorin Stancu-Mara, București 100
84. Valentin Dinu, Pitești 100
84. Valentin Gușă, București 100
84. Vlad Bălan, Bacău 100

Bursele Agora

Vă prezentăm în continuare enunțurile problemelor propuse spre rezolvare la etapele #09-#12 ale ediției 2002/2003 a concursului de programare Bursele Agora.

P010304: Piramidă

Se consideră o piramidă triunghiulară (determinată de patru puncte necoplanare din spațiu) și alte N puncte din spațiu. Să se determine câte dintre aceste N puncte se află în interiorul piramidei. Un punct este considerat a fi în interiorul piramidei chiar dacă se află pe una dintre cele patru fețe, pe una dintre cele șase muchii sau este unul dintre cele patru vârfuri ale piramidei.

Date de intrare

Primele patru linii ale fișierului de intrare `PYRAMID.IN` conțin câte trei numere întregi reprezentând coordonatele vârfurilor piramidei. Următoarea linie conține numărul N al celorlalte puncte. Fiecare dintre următoarele N linii va conține câte trei numere întregi reprezentând coordonatele unui punct.

Date de ieșire

Fișierul de ieșire `PYRAMID.OUT` trebuie să conțină o singură linie pe care se va afla numărul punctelor din interiorul piramidei.

Restricții și precizări

- $1 \le N \le 100000$;
- coordonatele punctelor sunt numere naturale cuprinse între 0 și 255;
- coordonatele tuturor punctelor sunt date în ordinea: coordonata pe axa Ox, coordonata pe axa Oy, coordonata pe axa Oz;

Exemplu

PYRAMID.IN	PYRAMID.OUT
0 0 0 0 0 100 0 100 0 100 0 0 2 100 100 100 1 1 1	1

Timp de execuție: 1 secundă/test

P010305: Poarta Stelară

Apophis dorește să viziteze toate cele N planete pe care le stăpânește. Oricare două dintre acestea sunt conectate prin intermediul unor porți stelare. El cunoaște care este cantitatea de energie necesară deplasării de la o planetă la alta și dorește să viziteze toate planetele consumând cât mai puțină energie.

Planetele aflate sub dominația lui *Apophis* sunt numerotate folosind numere întregi cuprinse între 1 și N. Inițial el se află pe planeta sa de reședință (identificată prin 1) și, după vizitarea tuturor planetelor el trebuie să se întoarcă aici.

Pentru ca locuitorii nici uneia dintre planete să nu se simtă favorizați, *Apophis* a decis că va vizita fiecare planetă (cu excepția celei de reședință) exact o dată.

Date de intrare

Prima linie a fișierului de intrare `STARGATE.IN` conține numărul N al planetelor care îl recunosc pe *Apophis* ca zeu.

Pe următoarele linii, până la sfârșitul fișierului, se află câte trei numere întregi x, y și c cu semnificația: *pentru deplasarea între planetele x și y este necesară o cantitate de energie c.*

Date de ieșire

Prima linie a fișierului de ieșire `STARGATE.OUT` va conține cantitatea totală de energie de care are nevoie *Apophis* pentru a vizita (pornind de pe planeta de reședință) toate planetele pe care le stăpânește și pentru a reveni pe planeta de reședință.

Pe următoarea linie se vor afla numerele de ordine ale planetelor (separate prin spații), în ordinea în care vor fi vizitate.

Restricții

- $3 \le N \le 500$;
- cantitatea de energie necesară trecerii de la o planetă la alta este un număr întreg cuprins între 1 și 255;
- cantitatea de energie necesară trecerii de la planeta x la planeta y este egală cu cea necesară trecerii de la planeta y la planeta x.

Exemplu

STARGATE.IN

```
4
1 2 2
1 3 3
1 4 4
2 3 6
2 4 8
3 4 12
```

STARGATE.OUT

```
21
1 3 2 4 1
```

Modalitatea de acordare a punctajului

Vom considera că pentru fiecare test se vor putea obține cel mult *X* puncte. Concurenții care vor obține cea mică valoare *CMin* pentru cantitatea totală de energie necesară vor primi *X* puncte pentru testul respectiv.

Ceilalți concurenți, care au furnizat un traseu valid pentru care este necesară o cantitate de energie *C*, vor obține $X \cdot CMin / C$ puncte pentru testul respectiv. Această valoare va fi aproximată cu două zecimale exacte.

Punctajul final va fi obținut prin adunarea punctajelor de la fiecare test și rotunjirea acestuia la cel mai apropiat număr întreg.

Dacă traseul nu este valid, concurenții nu vor primi nici un punct pentru testul respectiv.

De exemplu, dacă pentru un test se pot obține cel mult 5 puncte, cel mai bun rezultat obținut de un concurent constă într-un traseu pentru care cantitatea necesară de energie este 89, iar un alt concurent obține o soluție pentru care cantitatea necesară de energie este 145, atunci, pentru testul respectiv, punctajul concurentului va fi de $5 \cdot 89 / 145 = 3{,}06$ puncte.

Timp de execuție: 1 secundă/test

P010306: Ordine

Se consideră un șir de numere naturale cuprinse între 0 și 1000000. Va trebui să determinați lungimea celui mai lung subșir strict crescător al șirului dat.

Date de intrare

Aceasta este o problemă cu fișierele de intrare deschise. La adresa **http://www.ginfo.ro/concurs/runda11/index.shtml** se află un *link* spre o arhivă care conține cele zece fișiere de intrare. Numele acestora au forma **ORDERXX.IN**, unde **XX** ia valori între **01** și **10** și reprezintă numărul testului.

Fiecare dintre cele zece fișiere de intrare va conține o singură linie pe care se vor afla elementele șirului, separate prin spații.

Date de ieșire

Pentru această problemă nu va trebui să trimiteți un program care să o rezolve, ci doar cele zece fișiere de ieșire corecte. Acestea vor fi denumite **ORDERXX.OUT**, unde **XX** ia valori între **01** și **10** și reprezintă numărul testului.

Prima linie a fișierului de ieșire va conține textul `ORDER - TEST #XX`, unde `XX` reprezintă numărul testului. Cea de-a doua linie va conține un singur număr care reprezintă lungimea celui mai lung subșir strict crecător.

Exemplu

Vom considera că acest exemplu reprezintă testul **00**.

ORDER00.IN

```
1 3 2 5 4 8 8 9
```

ORDER00.OUT

```
ORDER - TEST #00
5
```

Modalitatea de acordare a punctajului

Pentru fiecare fișier de ieșire corect veți obține 10 puncte. Punctajul maxim care poate fi obținut (dacă toate cele 10 fișiere de ieșire sunt corecte) este de 100 de punte.

Trimiterea fișierelor de ieșire

Arhiva pe care o veți trimite se va numi **YYYYYR11.ZIP**, **YYYYYR11.RAR** sau **YYYYYR11.ACE** (în funcție de programul de arhivare pe care îl folosiți), unde **YYYYY** este codul dumneavoastră de identificare (ID).

Ea nu va conține fișiere *batch* (**YYYYYR11.BAT**) sau surse (**YYYYYR11.PAS**, **YYYYYR11.CPP** sau **YYYYYR11.C**). În arhivă se vor afla cele zece fișiere de ieșire corecte, denumirea lor fiind **ORDERXX.OUT**, unde **XX** reprezintă numărul testului (cuprins între **01** și **10**).

P080207: Sortare

Se consideră un șir *a* cu *N* elemente ale căror valori sunt numere întregi disticte care nu sunt cunoscute.

Va trebui să determinați o permutare *p* a mulțimii {1, 2, ..., *N*} astfel încât șirul *b* ale cărui elemente sunt date de formula $b[i] = a[p[i]]$ ($i = 1, ..., N$) să fie sortat în ordine crescătoare.

Pentru determinarea permutării *p* puteți efectua un număr limitat de comparări între două elementele ale șirului *a*.

Pentru aceasta aveți la dispoziție o bibliotecă externă care vă pune la dispoziție rutine pentru:

- inițializare;
- determinarea numărului de elemente ale șirului;
- determinarea numărului maxim de comparări pe care le aveți la dispoziție;
- compararea a două elemente ale șirului *a*;
- furnizarea ca rezultat a unui element al permutării *p*.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul dumneavoastră istrucțiunea **uses** `sort`; pentru limbajul *Pascal* și `#include "sort.h"` pentru limbajul *C/C++*.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

procedure `Init;`
void `Init()`
- trebuie apelată la începutul programului și este folosită de către bibliotecă pentru inițializare;
- întrerupe execuția programului dacă este apelată a doua oară.

function `GetN:Integer;`
int `GetN()`
- returnează numărul de elemente (N) ale șirului a (și ale permutării p care trebuie determinată).

function `GetNoComp:Integer;`
int `GetNoComp()`
- returnează numărul de comparări (apeluri ale funcției `IsGreater`) pe care le mai aveți la dispoziție (în momentul apelului funcției `GetNoComp`) pentru determinarea permutării p.

function `IsGreater(i,j:Integer):Boolean;`
int `IsGreater(int i,int j)`
- returnează `true` (valoare nenulă) dacă $a[i] > a[j]$ și `false` (0) în caz contrar;
- întrerupe execuția programului dacă cel puțin una dintre valorile `i` și `j` nu este un număr întreg cuprins între 1 și N sau dacă numărul de apeluri ale acestei funcții depășește numărul maxim permis.

procedure `Result(i,pi:Integer);`
void `Result(int i,int pi)`
- acceptă ca rezultat faptul că al `i`-lea element al permutării p este `pi`;
- întrerupe execuția programului dacă:
  - valoarea `i` nu reprezintă un număr întreg cuprins între 1 și N;
  - valoarea `pi` nu reprezintă un număr întreg cuprins între 1 și N;
  - valoarea celui de-al `i`-lea element al permutării p este furnizată a doua oară;
  - valoarea corectă a celui de-al `i`-lea element al permutării p nu este `pi`;
  - au fost furnizate valorile tuturor celor N elemente ale permutării (*Acesta este singurul caz în care sunt acordate punctele!*).

Toate funcțiile și procedurile (excepție: `Init`) întrerup execuția programului dacă sunt apelate înaintea inițializării.

Restricții și precizări

- Programul vostru nu va citi date din nici un fișier de intrare și nu va scrie date în nici un fișier de ieșire.
- $2 \le N \le 255$.

Exemple de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
`Init`	-
`GetN`	3
`GetMax`	3
`IsGreater(1,2)`	`true (1)`
`IsGreater(1,3)`	`false (0)`
`IsGreater(2,3)`	`false (0)`
`Result(1,2)`	-
`Result(2,1)`	-
`Result(3,3)`	-

Subprogram apelat	*Valoare returnată*
`Init`	-
`GetN`	5
`GetMax`	10
`IsGreater(1,2)`	`true (1)`
`IsGreater(1,3)`	`true (1)`
`IsGreater(1,4)`	`true (1)`
`IsGreater(1,5)`	`true (1)`
`IsGreater(2,3)`	`true (1)`
`IsGreater(2,4)`	`true (1)`
`IsGreater(2,5)`	`true (1)`
`IsGreater(3,4)`	`true (1)`
`IsGreater(3,5)`	`true (1)`
`IsGreater(4,5)`	`true (1)`
`Result(1,10)`	-
`Result(2,9)`	-
`Result(3,8)`	-
`Result(4,7)`	-
`Result(5,6)`	-
`Result(6,5)`	-
`Result(7,4)`	-
`Result(8,3)`	-
`Result(9,2)`	-
`Result(10,1)`	-

Subprogram apelat	*Valoare returnată*
`Init`	-
`GetN`	5
`GetMax`	8
`IsGreater(1,2)`	`false (0)`
`IsGreater(3,4)`	`false (0)`
`IsGreater(4,5)`	`false (0)`
`IsGreater(3,5)`	`false (0)`
`IsGreater(1,3)`	`false (0)`
`IsGreater(2,3)`	`true (1)`
`IsGreater(2,4)`	`true (1)`
`IsGreater(2,5)`	`false (0)`
`Result(1,1)`	-
`Result(2,3)`	-
`Result(3,4)`	-
`Result(4,2)`	-
`Result(5,5)`	-

Timp de execuție: 1 secundă/test

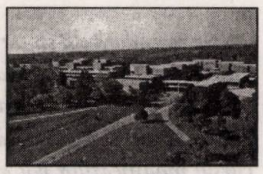
OLIMPIADELE internaționale din 2003

Mihai Scorțaru

Cele mai importante concursuri internaționale de programare la care participă elevii români sunt Olimpiada Internațională de Informatică, Olimpiada de Informatică a Europei Centrale și Olimpiada Balcanică de Informatică. În cadrul acestui articol vă vom prezenta câteva detalii referitoarea la edițiile din acest an ale acestor concursuri.

IOI 2003

Cea de-a XV-a ediție a *Olimpiadei Internaționale de Informatică* se va desfășura în perioada 16-23 august 2003 la Universitatea *Wisconsin-Parkside* din *Statele Unite ale Americii*.



Universitatea Wisconsin

Kenosha, orașul în care se va desfășura olimpiada, se află în statul american *Wisconsin*, la nord de *Chicago*, în apropierea *Lacului Michigan*.

Pentru petrecerea timpului liber, concurenții, ceilalți membri ai delegațiilor și invitații vor putea beneficia de facilitățile din campus: bazine pentru înot, terenuri de fotbal și golf, săli de popice și de biliard etc.

Ca de obicei, vor fi două zile de concurs (18 și 21 august), iar premierea va avea loc pe data de 22 august. În zilele de 19 și 20 august se vor organiza excursii în *Chicago* și *Milwaukee*.

Fiecare țară participantă va fi reprezentată de cel mult patru concurenți care vor fi însoțiți de un *team leader* și un *deputy leader*. Pe lângă cei cel mult șase reprezentanți ai fiecărei țări, la *IOI* 2003 vor participa diferiți invitați, precum și membrii comitetelor de organizare (*IC - International Committee* și *ISC - International Scientific Committee*).

La fel ca la ultimele două ediții, limbajele de programare pe care le vor folosi concurenții sunt *FreePascal* și *GNU C++*. Pe calculatoarele concurenților vor fi instalate sisteme de operare *Linux* și *Windows*, dar evaluarea se va realiza folosind *Linux*.

Site-ul oficial al ediției din acest an a *Olimpiadei Internaționale de Informatică* (*IOI* 2003) poate fi accesat la adresa **http://www.ioi2003.org**. Aici veți găsi informații actualizate permanent referitoare la această ediție a concursului. Din nefericire, până la închiderea ediției pe majoritatea paginilor apărea textul "*To be added later*".

Informații legate de edițiile anterioare ale *Olimpiadei Internaționale de Informatică* pot fi accesate la adresele **http://olympiads.win.tue.nl/ioi** și **http://www.ioinformatics.org**.

CEOI 2003

Olimpiada de Informatică a Europei Centrale a fost inițiată de țara noastră în anul 1994. Până acum *România* a organizat edițiile din anii 1994 și 2000 ale acestei competiții.

Cea de-a X-a ediție se va desfășura în localitatea germană *Münster* și organizatorii promit că nu vor pierde ocazia de a sărbători această ediție jubiliară.

Așadar, cu puțin peste o lună înaintea începerii *Olimpiadei Internaționale de Informatică* (mai exact în perioada 5-12 iulie 2003).

Din nefericire, *site*-ul dedicat acestui concurs conține foarte puține informații. Totuși, aici veți găsi în lunile care urmează mai multe detalii referitoare la *CEOI* 2003. *Site*-ul poate fi accesat la adresa **http://www.bwinf.de/ceoi2003**.

BOI 2003

Din câte știm, *Balcaniada de Informatică* din acest an ar trebui să se desfășoare în *România*, la *Sinaia*.

Din nefericire, aceasta este singura informație pe care o deținem despre ediția din acest an a acestui concurs.

La închiderea ediției nu exista nici un *site Web* dedicat celei de-a XI-a ediții a *Olimpiadei Balcanice de Informatică*.

ARBORI indexați BINAR

Mihai Scorțaru

Problema determinării sumei elementelor unei subsecvențe a unui șir ale cărui valori se modifică în timp real apare destul de des în diferite aplicații. Ea a apărut, sub diverse forme și la anumite concursuri de programare. În cadrul acestui articol vom prezenta o structură de date care poate fi folosită pentru rezolvarea eficientă a acestei probleme.

Introducere

Prin ***subsecvență*** înțelegem un subșir ale cărui elemente se află pe poziții consecutive în șirul inițial. De exemplu, (2, 3, 4) este o subsecvență a șirului (1, 2, 3, 4, 5) formată din al doilea, al treilea și al patrulea element al șirului, în timp ce (1, 2, 4) nu este o subsecvență deoarece nu este formată din elemente aflate pe poziții consecutive.

În cadrul acestui articol vom identifica o subsecvență prin extremitățile sale (poziția cea mai din stânga și poziția cea mai din dreapta din șir). O subsecvență care începe în poziția a și se termină în poziția b va fi notată prin $<a, b>$. Pentru șirul (1, 2, 3, 4, 5) subsecvența (2, 3, 4) va fi notată prin <2, 4> dacă numerotarea elementelor începe cu 1 sau prin <1, 3> dacă numerotarea începe de la 0.

Deseori, avem nevoie de informații referitoare la subsecvențele unui șir cum ar fi suma elementelor, produsul lor, valoarea minimă, valoarea maximă etc. La prima vedere, rezolvarea acestei probleme pare destul de simplă (de exemplu, pentru sumă se parcurg elementele subsecvenței și se adună). Totuși, acest algoritm este ineficient datorită faptului că necesită parcurgerea întregii subsecvențe. Vom arăta că există algoritmi pentru care o astfel de parcurgere nu este necesară.

Totuși, dacă am efectua o singură dată sau de un număr limitat de ori o astfel de parcurgere, algoritmul ar putea părea performant, viteza de execuție fiind relativ mare. Problema se complică dacă elementele șirului se modifică în timp real.

Modificări în timp real

Să presupunem că există două tipuri de operații care pot fi efectuate asupra unui șir. Primul tip constă în modificarea valorii unui element, în timp ce al doilea reprezintă interogări (cereri de informații) referitoare la anumite subsecvențe ale șirului. De obicei, cele două tipuri de operații sunt "amestecate" în sensul că nu vor fi doar modificări urmate din interogări, ci putem avea o modificare, urmată de două interogări, urmate de cinci modificări, urmate de alte două interogări etc.

Așadar, putem spune că elementele șirului se modifică în timp real și interogările se referă la starea curentă a șirului (cea din momentul cererii de informații).

Enunțul problemei

În cele ce urmează vom prezenta un enunț al problemei, particularizat pentru cazul în care interogările se referă la suma elementelor subsecvențelor.

Se consideră un șir de numere întregi care are toate elementele nule. O modificare a unui element constă în adunarea unei valori la valoarea curentă (pot fi realizate și scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unei subsecvențe a șirului.

Problema poate fi enunțată în multe alte forme. Una dintre ele ar putea fi următoarea:

Un furnizor lucrează cu N distribuitori; în fiecare moment distribuitorii pot efectua plăți sau pot cumpăra produse. De asemenea, în fiecare moment furnizorul poate cere informații referitoare la suma totală a datoriilor pe care le au magazinele cu numere de ordine cuprinse între două valori date.

Evident, există multe alte enunțuri echivalente. De asemenea, pot apărea enunțuri în care operația de însumare ar putea fi înlocuită cu altele. De exemplu, furnizorul ar putea dori să cunoască datoria maximă a unui magazin care are numărul de ordine cuprins între două valori date.

Exemplu

Vom exemplifica acum evoluția în timp real a unui șir format din cinci elemente, prezentând valorile șirului după fiecare modificare și rezultatul fiecărei interogări.

Operația **Inițializare** constă în setarea la 0 a valorilor tuturor elementelor. Operația **Adună**(i, x) constă în adunarea valorii x la valoarea curentă a celui de-al i-lea element.

Operația Sumă(a, b) furnizează suma elementelor subsecvenței <a, b>.

Operație	*Șir/Rezultat*
Inițializare	0 0 0 0 0
Adună(1, 3)	3 0 0 0 0
Adună(4, 5)	3 0 0 5 0
Sumă(3, 3)	0
Sumă(4, 4)	5
Adună(4, 2)	3 0 0 7 0
Adună(2, 3)	3 3 0 7 0
Sumă(2, 5)	10
Sumă(2, 4)	10
Adună(5, 2)	3 3 0 7 2
Sumă(2, 4)	10
Sumă(2, 5)	12
Adună(3, 1)	3 3 1 7 2
Adună(4, -2)	3 3 1 5 2
Sumă(2, 5)	11
Adună(1, -3)	0 3 1 5 2
Adună(5, 5)	0 3 1 5 7
Sumă(1, 2)	3
Sumă(3, 4)	6
Adună(2, -1)	0 2 1 5 7
Adună(3, 3)	0 2 4 5 7
Sumă(5, 5)	7
Sumă(1, 2)	2
Sumă(1, 5)	18

Cazul unidimensional

Din modul în care a fost enunțată problema, rezultă că operațiile sunt efectuate asupra unui tablou unidimensional; așadar, acesta este cazul unidimensional al problemei. Vom prezenta în continuare trei algoritmi care pot fi folosiți pentru rezolvarea problemei și apoi le vom studia performanțele.

Algoritmul naiv

Cea mai intuitivă metodă de rezolvare constă în păstrarea unui vector cu valorile șirului, modificarea lor atunci când este necesar și calcularea sumelor în momentul în care apar interogări.

Pentru a prezenta algoritmul vom considera că o modificare este codificată prin valoarea 1, iar o interogare prin valoarea 2. Ar fi necesară o a treia operație (codificată prin 3) care ar indica faptul că nu mai există modificări sau interogări, deci execuția poate fi încheiată.

Versiunea în pseudocod este prezentată în continuare:

```
//inițializări
scrie Introduceți numărul de elemente:
citește N      //dimensiunea șirului
pentru i ← 1, N execută
   a_i ← 0     //valorile inițiale sunt nule
sfârșit pentru
scrie Introduceți codul operației:
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1  //modificare
    atunci
      scrie Introduceți indicele elementului care va fi
            modificat:
      citește ind
      scrie Introduceți valoarea care va fi adunată
            (valoare negativă pentru scăderi):
      citește val
      a_ind ← a_ind + val
    altfel //interogare
      scrie Introduceți extremitățile subsecvenței:
      citește st, dr
      suma ← 0
      pentru i ← st, dr execută
        suma ← suma + a_i
      sfârșit pentru
      scrie Suma elementelor secvenței este, suma
  sfârșit dacă
  scrie Introduceți codul operației:
  citește cod
sfârșit cât timp
```

Se observă că modificările se efectuează în timp constant deoarece implică doar accesarea unui element al vectorului și modificarea valorii sale.

Datorită faptului că pentru calcularea sumei elementelor unei subsecvențe este necesară parcurgerea tuturor elementelor subsecvenței, această operație are ordinul de complexitate $O(l)$, unde l este lungimea subsecvenței.

Trebuie observat faptul că algoritmul este același dacă operația de însumare este înlocuită cu o alta (calcularea produsului, stabilirea minimului etc.).

Vector de sume

Prima încercare de optimizare constă în găsirea unui algoritm mai rapid pentru calcularea sumei elementelor unei subsecvențe. O posibilitate relativ simplă este păstrarea unui vector b a cărui elemente b_i reprezintă suma primelor elemente ale șirului a. Pentru a găsi suma elementelor unei subsecvențe <st, dr> vom efectua o simplă diferență: $b_{dr} - b_{st-1}$.

Pentru a calcula sumele pentru subsecvențe de forma <1, dr> vom avea nevoie de elementul b_0 care va avea întotdeauna valoarea 0. Astfel, pentru o subsecvență de această formă suma elementelor va fi $b_{dr} - b_0 = b_{dr}$.

Așadar, folosind acest artificiu, ordinul de complexitate al unei interogări va fi $O(1)$ pentru că este necesară doar o simplă scădere pentru furnizarea rezultatului. Din nefericire, în momentul efectuării unei modificări pentru elementul i, toate elementele b_j pentru care $j \geq i$ își modifică valoarea.

Ca urmare, operația de modificare a celui de-al i-lea element nu se mai realizează în timp constant, deoarece trebuie modificate $N - i + 1$ valori. Așadar, ordinul de complexitate devine liniar.

Astfel, am reușit să înlocuim algoritmul liniar de determinare a sumei elementelor unei subsecvențe cu un algoritm având ordinul de complexitate $O(1)$ cu prețul creșterii timpului de execuție a algoritmului de modificare de la unul constant la unul liniar.

Se observă că nu mai avem nevoie de șirul *a* folosit pentru algoritmul anterior, fiind suficientă păstrarea valorilor elementelor șirului *b*. Prezentăm versiunea în pseudocod a acestui algoritm, folosind aceleași coduri pentru operațiile efectuate:

```
//inițializări
scrie Introduceți numărul de elemente:
citește N      //dimensiunea șirului
pentru i ← 0, N execută
  b_i ← 0        //valorile inițiale sunt nule
sfârșit pentru
scrie Introduceți codul operației:
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1 //modificare
    atunci
      scrie Introduceți indicele elementului care va fi
            modificat:
      citește ind
      scrie Introduceți valoarea care va fi adunată
            (valoare negativă pentru scăderi):
      citește val
      pentru i ← ind, N execută
        b_i ← b_i + val
      sfârșit pentru
    altfel //interogare
      scrie Introduceți extremitățile subsecvenței:
      citește st, dr
      scrie Suma elementelor secvenței este b_dr - b_st-1
  sfârșit dacă
  scrie Introduceți codul operației:
  citește cod
sfârșit cât timp
```

Din nou, algoritmul este același dacă operația de însumare este înlocuită cu o alta (calcularea produsului, stabilirea minimului etc.).

Arbore indexat binar

Practic, pentru algoritmii anteriori una dintre cele două operații se efectuează în timp constant, în timp ce cealaltă se efectuează în timp liniar. Așadar, dacă numărul de modificări este aproximativ egal cu cel al interogărilor, timpul de execuție va fi aproximativ același.

Dacă numărul modificărilor este mult mai mare decât cel al interogărilor, atunci este preferabilă folosirea algoritmului naiv.

Dacă, dimpotrivă, numărul interogărilor este mult mai mare decât cel al modificărilor, atunci algoritmul bazat pe vectorul de sume este mai performant. Totuși, în cazul mediu, ambii algoritmi sunt liniari.

În cele ce urmează vom prezenta o structură de date care permite efectuarea în timp logaritmic atât a modificărilor, cât și a interogărilor. Structura de date pe care o propunem este numită ***arbore indexat binar***.

Așadar, vom avea o structură arborescentă care va permite efectuarea interogărilor în timp logaritmic în condițiile în care și modificările se efectuează în timp logaritmic.

Pentru a folosi această structură de date, va trebui să considerăm că elementele șirului sunt numerotate începând cu 1.

Arborele va fi păstrat sub forma unui vector *c* în care fiecare element *i* va conține suma elementelor subsecvenței $<i - 2^k + 1, i>$, unde *k* este numărul zerourilor terminale din reprezentarea binară a lui *i*.

Așadar, elementele de pe pozițiile impare ale arborelui vor păstra suma elementelor unor subsecvențe formate dintr-un singur element (aflat pe o poziție impară în șirul *a*). În elementele de pe pozițiile de forma $4 \cdot k + 2$ (un zero terminal în reprezentarea binară a poziției) vom păstra suma elementelor unor subsecvențe formate din două elemente. În elementele de pe pozițiile de forma $8 \cdot k + 4$ (două zerouri terminale în reprezentarea binară a poziției) vom păstra suma elementelor unor subsecvențe formate din patru elemente.

În general, dacă reprezentarea binară a pozițiilor au *p* zerouri terminale, atunci elementele din arbore vor păstra sume ale elementelor unor subsecvențe cu 2^p elemente.

Vom considera că, la un moment dat, șirul (format din 15 elemente) este (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15).

Valorile vectorului *c* (cel care reprezintă arborele indexat binar) sunt:

Element	*Poziție*	*Semnificație*	*Valoare*
c_1	0001	Sumă(1, 1)	1
c_2	0010	Sumă(1, 2)	3
c_3	0011	Sumă(3, 3)	3
c_4	0100	Sumă(1, 4)	10
c_5	0101	Sumă(5, 5)	5
c_6	0110	Sumă(5, 6)	11
c_7	0111	Sumă(7, 7)	7
c_8	1000	Sumă(1, 8)	36
c_9	1001	Sumă(9, 9)	9
c_{10}	1010	Sumă(9, 10)	19
c_{11}	1011	Sumă(11, 11)	11
c_{12}	1100	Sumă(9, 12)	42
c_{13}	1101	Sumă(13, 13)	13
c_{14}	1110	Sumă(13, 14)	27
c_{15}	1111	Sumă(15, 15)	15

Vom prezenta în continuare modul în care se efectuează modificările, exemplificând pe arborele prezentat anterior. În acest scop, vom modifica valoarea celui de-al doilea element din 2 în 8 (se adună valoarea 6).

Se observă că trebuie modificate toate valorile vectorului *c* care reprezintă sume ale unei subsecvențe care conține al doilea element al șirului; acestea sunt: c_2, c_4 și c_8. Reprezentările binare ale acestor trei poziții sunt 0010, 0100 și 1000.

Se observă că, fiecare astfel de reprezentare (evident, cu excepția primeia) poate fi obținută din cea anterioară prin adunarea valorii 2^r, unde *r* reprezintă numărul de zerouri terminale ale reprezentării binare a poziției anterioare.

Așadar, ar trebui să efectuăm astfel de adunări până în momentul în care poziția obținută este mai mare decât dimensiunea șirului și să creștem cu 6 valorile tuturor elementelor de pe pozițiile de la fiecare pas.

Vom exemplifica procedeul și pentru al treilea element. Prima poziție este 0011; numărul zerourilor terminale este 0, deci vom aduna valoarea $2^0 = 1$. Noua poziție va fi 3 + 1 = 4, deci am ajuns la poziția 4 a cărei reprezentare binară este 0100. Numărul zerourilor terminale este 2, deci vom aduna valoarea $2^2 = 4$, poziția devenind 4 + 4 = 8. Reprezentarea binară este 1000, deci avem trei zerouri terminale. Valoarea adunată va fi $2^3 = 8$, deci ajungem în poziția 8 + 8 = 16, așadar am depășit dimensiunea șirului.

Se observă că elementele c_3, c_4 și c_8 sunt cele a căror valoare depinde de valoarea elementului de pe a treia poziție.

Algoritmul care realizează operația de modificare este următorul:

- Se identifică poziția elementului care trebuie modificat.
- Cât timp poziția curentă este cel mult egală cu dimensiunea șirului:
  - Se modifică valoarea elementului de pe poziția curentă.
  - Se determină numărul *k* al zerourilor terminale din reprezentarea binară a poziției curente.
  - Noua poziție se determină adunând valoarea 2^k la poziția curentă.

Se observă că numărul elementelor modificate este cel mult egal cu numărul cifrelor reprezentării binare a numărului de elemente din șir, deci operația are ordinul de complexitate $O(\log N)$.

Mai trebuie prezentat modul în care este efectuată operația de interogare. Vom încerca să aplicăm o metodă similară celei propuse în cazul algoritmului care folosește un vector de sume. Pentru aceasta, dacă dorim să determinăm suma elementelor subsecvenței <*st*, *dr*>, vom determina sumele elementelor subsecvențelor <1, *st* - 1> și <1, *dr*>, efectuând apoi o simplă diferență.

Operațiile efectuate sunt, oarecum, inversele celor de la operația de modificare. Vom porni din poziția dată și, la fiecare pas, vom determina următoarea poziție scăzând valoarea 2^k unde *k* reprezintă numărul zerourilor terminale din reprezentarea binară a poziției curente. Ne vom opri în momentul în care poziția curentă devine 0.

De exemplu, pentru a determina suma elementelor subsecvenței <1, 11> vom porni din poziția 11 a cărei reprezentare binară este 1011. Numărul zerourilor terminale este 0, deci vom scădea valoarea $2^0 = 1$, ajungând în poziția 11 - 1 = 10. Reprezentarea binară a acesteia este 1010, deci numărul zerourilor terminale este 1. Valoarea scăzută este $2^1 = 2$, deci se ajunge în poziția 10 - 2 = 8, a cărei reprezentare binară este 1000. De data aceasta avem trei zerouri terminale, deci vom scădea valoarea $2^3 = 8$ ajungând în poziția 8 - 8 = 0.

Așadar, am "trecut" prin pozițiile 11, 10 și 8. Adunând valorile elementelor corespunzătoare (c_{11}, c_{10} și c_8) obținem 11 + 19 + 36 = 66, adică exact valoarea căutată.

Este ușor de observat că c_{11} reprezintă suma elementelor subsecvenței <11, 11>, c_{10} reprezintă suma elementelor subsecvenței <9, 10>, iar c_8 reprezintă suma elementelor subsecvenței <1, 8>, așadar, în momentul calculării sumei, fiecare element de pe o poziție mai mică sau egală cu 11 este adunat o singură dată.

Pentru a determina suma elementelor unei subsecvențe de forma <1, *dr*> vom folosi următorul algoritm:

- Se identifică elementul de pe poziția *dr*.
- Cât timp poziția curentă este diferită de 0:
  - Se adună la suma totală valoarea elementului de pe poziția curentă.
  - Se determină numărul *k* al zerourilor terminale din reprezentarea binară a poziției curente.
  - Noua poziție se determină scăzând valoarea 2^k din poziția curentă.

Evident, pentru a determina suma elementelor unei subsecvențe de forma <*st*, *dr*> vom aplica de două ori algoritmul prezentat: o dată pentru subsecvența <1, *dr*> și o dată pentru subsecvența <1, *st*-1>, efectuând la final diferența valorilor obținute.

De exemplu, pentru determinarea sumei elementelor subsecvenței <4, 13> vom calcula mai întâi sumele elementelor subsecvențelor <1, 13> și <1, 3>. Acestea sunt **Sumă**(1, 13) = $c_{13} + c_{12} + c_8 = 13 + 42 + 36 = 91$ și **Sumă**(1, 3) = $c_3 + c_2 = 3 + 3 = 6$, diferența lor fiind 91 - 6 = 85 care este egală cu valoarea **Sumă**(4, 13).

Se observă că numărul elementelor adunate este cel mult egal cu numărul cifrelor reprezentării binare a poziției elementului dat, deci operația are ordinul de complexitate $O(\log N)$.

Din nou, se observă că nu avem nevoie de păstrarea elementelor șirului *a*, fiind suficientă păstrarea valorilor vectorului *c* (cel care reprezintă arborele indexat binar).

Varianta în pseudocod a algoritmului de rezolvare a problemei folosind un arbore indexat binar este următoarea:

```
//inițializări
scrie Introduceți numărul de elemente:
citește N      //dimensiunea șirului
pentru i ← 0, N execută
   c_i ← 0     //valorile inițiale sunt nule
sfârșit pentru
```

```
scrie Introduceți codul operației:
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1 //modificare
    atunci
      scrie Introduceți indicele elementului care va fi
            modificat:
      citește ind
      scrie Introduceți valoarea care va fi adunată
            (valoare negativă pentru scăderi):
      citește val
      poz ← 0  //poziția celui mai nesemnficativ bit cu
               //valoarea 1
      cât timp ind ≤ N execută
        c_ind ← c_ind + val
        cât timp ind & 2^poz ← 0  execută
                 //& reprezintă operația de conjucție
                 //logică (ȘI-logic)
          poz ← poz + 1
        sfârșit cât timp
        ind ← ind + 2^poz
        poz ← poz + 1
                 //valoarea poz este incrementată
                 //și nu reinițializată cu 0 deoarece
                 //știm că acum avem cel puțin
                 //poz + 1 zeroruri terminale, prin
                 //adunare adăugându-se cel puțin
                 //un zero terminal
      sfârșit cât timp
    altfel //interogare
      scrie Introduceți extremitățile subsecvenței:
      citește st, dr
      //calcularea primei sume
      s1 ← 0    //inițializare
      poz ← 0  //poziția celui mai nesemnficativ bit cu
               //valoarea 1
      cât timp dr > 0 execută
        s1 ← s1 + c_dr
        cât timp dr & 2^poz = 0 execută
          poz ← poz + 1
        sfârșit cât timp
        dr ← dr - 2^poz
        poz ← poz + 1  //prin scădere se adaugă cel
                       //puțin un zero terminal
      sfârșit cât timp
      //calcularea celei de-a doua sume
      st ← st - 1  //trebuie calculată valoarea pentru
                   //st-1
      s2 ← 0     //inițializare
      poz ← 0  //poziția celui mai nesemnficativ bit cu
               //valoarea 1
      cât timp st > 0 execută
        s2 ← s2 + c_st
        cât timp st & 2^poz = 0 execută
          poz ← poz + 1
        sfârșit cât timp
        st ← st - 2^poz
        poz ← poz + 1
      sfârșit cât timp
      scrie Suma elementelor subsecvenței este: s1 -
            s2
  sfârșit dacă
  scrie Introduceți codul operației:
  citește cod
sfârșit cât timp
```

Și de data aceasta putem înlocui operația de însumare cu alte operații (înmulțire, minim, maxim etc.).

Cazul bidimensional

Problema enunțată poate fi modificată astfel încât să lucrăm în spațiul bidimensional. Enunțul noii probleme ar putea fi următorul:

Se consideră o matrice de numere întregi care are toate elementele nule. O modificare a unui element al matricei constă în adunarea unei valori la valoarea curentă (pot fi realizate și scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unei submatrice.

Așadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă dreptunghiulară" care face parte din matrice.

Evident, pentru reprezentarea datelor vom folosi tablouri bidimensionale în locul celor unidimensionale (matrice în locul vectorilor).

Din nou, enunțul poate fi reformulat în diferite moduri.

O variantă ar putea fi:

Se consideră un teren de formă dreptunghiulară având lungimea M și lățimea N. Terenul este împărțit în regiuni de formă pătrată având latura egală cu unitatea. În fiecare moment, într-o anumită regiune pot fi plantați sau tăiați pomi. De asemenea, în fiecare moment proprietarul terenului ar putea cere informații referitoare la numărul total al pomilor care se află într-o regiune dreptunghiulară a terenului.

Vom adapta cei trei algoritmi descriși pentru cazul unidimensional pentru a rezolva noua problemă și vom studia apoi performanțele acestora.

De data aceasta, pentru operația de modificare vom avea nevoie de doi parametri care vor reprezenta linia și coloana pe care se află elementul a cărui valoare va fi modificată.

Pentru interogări vom avea nevoie de patru parametri care vor reprezenta coordonatele colțurilor din stânga-sus și dreapta-jos ale unui dreptunghi.

Vom exemplifica acum cazul bidimensional al problemei folosind o matrice cu trei linii și trei coloane.

Operație	*Matrice/Rezultat*
Inițializare	0 0 0
	0 0 0
	0 0 0
Adună(2, 2, 3)	0 0 0
	0 3 0
	0 0 0
Adună(1, 3, 5)	0 0 5
	0 3 0
	0 0 0
Sumă(2, 1, 3, 3)	3
Adună(2, 1, 1)	0 0 5
	1 3 0
	0 0 0
Sumă(1, 2, 2, 3)	8
Adună(2, 2, -1)	0 0 5
	1 2 0
	0 0 0
Sumă(1, 2, 2, 3)	7
Adună(2, 2, 5)	0 0 5
	1 7 0
	0 0 0
Adună(2, 2, -5)	0 0 5
	1 2 0
	0 0 0
Adună(3, 2, 4)	0 0 5
	1 2 0
	0 4 0
Sumă(1, 1, 3, 3)	12

Algoritmul naiv

Primul algoritm descris pentru cazul unidimensional poate fi adaptat foarte ușor pentru a rezolva cazul bidimensional al problemei. Vom păstra valorile matricei, le vom modifica dacă este necesar și vom calcula sumele cerute de interogări.

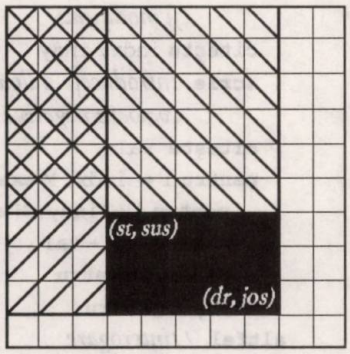
Versiunea în pseudocod este prezentată în continuare:

```
//inițializări
scrie Introduceți dimensiunile matricei:
citește M, N
pentru i ← 1, M execută
  pentru j ← 1, N execută
    a_ij ← 0        //valorile inițiale sunt nule
  sfârșit pentru
sfârșit pentru
scrie Introduceți codul operației:
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1 //modificare
    atunci
      scrie Introduceți indicii elementului care va fi
            modificat:
      citește indx, indy
      scrie Introduceți valoarea care va fi adunată
            (valoare negativă pentru scăderi):
      citește val
      a_indx,indy ← a_indx,indy + val
    altfel //interogare
      scrie Introduceți coordonatele colțurilor:
      citește st, sus, dr, jos
      suma ← 0
      pentru i ← sus, jos execută
        pentru j ← st, dr execută
          suma ← suma + a_ij
        sfârșit pentru
      sfârșit pentru
      scrie Suma elementelor dreptunghiului este: suma
  sfârșit dacă
  scrie Introduceți codul operației
  citește cod
sfârșit cât timp
```

Modificările se efectuează tot în timp constant deoarece implică doar accesarea unui element al matricei și modificarea valorii sale.

Datorită faptului că pentru calcularea sumei elementelor dintr-un dreptunghi este necesară parcurgerea tuturor elementelor dreptunghiului, această operație are ordinul de complexitate $O(l \cdot h)$, unde l și h sunt lungimile laturilor dreptunghiului.

Matrice de sume



Adaptarea algoritmului care folosește un vector de sume este destul de simplă. Vom păstra o matrice b ale cărei elemente b_{ij} vor conține sumele corespunzătoare dreptunghiurilor cu colțul din stânga-sus în poziția (1, 1) și cel din dreapta-jos în poziția (i, j). Studiind figura alăturată, este ușor de observat că pentru a calcula valoarea **Sumă**(*st*, *sus*, *dr*, *jos*) poate fi folosită relația:

Sumă(*st*, *sus*, *dr*, *jos*) = **Sumă**(1, 1, *dr*, *jos*) -
Sumă(1, 1, *st* - 1, *jos*) -
Sumă(1, 1, *dr*, *sus* - 1) +
Sumă(1, 1, *st* - 1, *sus* - 1).

Se observă că prin scăderea valorii **Sumă**(1, 1, *st*-1, *jos*) se scad valorile tuturor elementelor aflate la dreapta dreptunghiului considerat, iar prin scăderea valorii **Sumă**(1, 1, *dr*, *sus*-1) se scad valorile elementelor aflate deasupra dreptunghiului.

Aceste operații implică scăderea de două ori a tuturor elementelor aflate la stânga și deasupra dreptunghiului considerat, motiv pentru care va trebui să adunăm valoarea **Sumă**(1, 1, *st*-1, *sus*-1).

Folosind acest artificiu avem nevoie de accesarea a patru elemente ale matricei b, deci operația se realizează în timp constant.

Din nou, în momentul modificării valorii unui element de coordonate (x, y), va trebui să modificăm valorile tuturor elementelor matricei b de coordonate (i, j) pentru care $i \geq x$ și $j \geq y$. Așadar, vom modifica $(M - x + 1) \cdot (N - y + 1)$ elemente, ordinul de complexitate devenind $O(M \cdot N)$.

Pentru a putea determina valori pentru dreptghiuri în care colțul din stânga-sus are una dintre coordonate egală cu 1 va trebui să considerăm că valorile elementelor matricei sunt nule dacă cel puțin unul dintre cei doi indici este 0.

Prezentăm versiunea în pseudocod a algoritmului descris:

```
//inițializări
scrie Introduceți dimensiunile matricei:
citește M, N      //dimensiunea șirului
pentru i = 0, M execută
  pentru i = 0, N execută
    b_ij = 0      //valorile inițiale sunt nule
  sfârșit pentru
sfârșit pentru
scrie Introduceți codul operației:
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1  //modificare
    atunci
      scrie Introduceți indicii elementului care va
            fi modificat:
      citește indx, indy
      scrie Introduceți valoarea care va fi adunată
            (valoare negativă pentru scăderi):
      citește val
      pentru i ← indx, M execută
        pentru i ← indy, N execută
          b_ij ← b_ij + val
        sfârșit pentru
      sfârșit pentru
    altfel //interogare
      scrie Introduceți coordonatele colțurilor:
      citește st, sus, dr, jos
      scrie Suma elementelor secvenței este: b_dr,jos -
            b_st-1,jos - b_dr,sus-1 + b_st-1,sus-1.
  sfârșit dacă
  scrie Introduceți codul operației
  citește cod
sfârșit cât timp
```

Din nou, algoritmul este foare asemănător dacă operația de însumare este înlocuită cu o alta (calcularea produsului, stabilirea minimului etc.).

Arbore de arbori indexați binar

Din nou, avem doi algoritmi în care una dintre operații este eficientă, în timp ce cealaltă necesită un timp de execuție mai mare. Folosind arborii indexați binar, vom putea găsi algoritmi care realizează ambele operații într-un timp de ordinul $O(\log M \cdot \log N)$, unde M și N sunt dimensiunile matricei.

Pentru a rezolva problema vom crea un ***arbore de arbori indexați binar.***

Așadar, vom avea o matrice c ale cărei elemente c_{ij} vor reprezenta sumele elementelor din dreptunghiul care are colțul din stânga-sus în poziția $(i - 2^k + 1, j - 2^l + 1)$, iar cel din dreapta-jos în poziția (i, j). Valoarea k reprezintă numărul zerourilor terminale din reprezentarea binară a lui i, iar l reprezintă numărul zerourilor terminale din reprezentarea binară a lui j.

De exemplu, pentru matricea:

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
```

elementele arborelui de arbori indexați binar vor avea următoarele semnficații:

Sumă(1, 1, 1, 1) **Sumă**(1, 1, 1, 2) **Sumă**(1, 3, 1, 3) **Sumă**(1, 1, 1, 4)
Sumă(1, 1, 2, 1) **Sumă**(1, 1, 2, 2) **Sumă**(1, 3, 2, 3) **Sumă**(1, 1, 2, 4)
Sumă(3, 1, 3, 1) **Sumă**(3, 1, 3, 2) **Sumă**(3, 3, 3, 3) **Sumă**(3, 3, 3, 4)
Sumă(1, 1, 4, 1) **Sumă**(1, 1, 4, 2) **Sumă**(1, 3, 4, 3) **Sumă**(1, 1, 4, 4)

Așadar, valorile matricei care reprezintă arborele binar vor fi:

```
1  3  3  6
6  14 10 36
9  19 11 23
28 60 36 136
```

Să presupunem că trebuie să modificăm valoarea elementului de coordonate (2, 1) care face parte dintr-o matrice cu 8 linii și 4 coloane. Vom aplica de două ori principiul descris pentru cazul unidimensional.

Vom identifica linia i pe care se află elementul și vom efectua modificări în toate coloanele de pe acea linie în care acestea trebuie efectuate. Pentru aceasta vom identifica coloana j, vom determina numărul k al zerourilor terminale și apoi vom aduna la j valoarea 2^k. Ajungem într-o nouă poziție și continuăm procedeul până în momentul în care valoarea coloanei curente depășește numărul de coloane al matricei.

În acest moment vom determina numărul zerourilor terminale l din reprezentarea binară a valorii i și ne vom "muta" pe linia $i + 2^l$. Vom reveni la coloana j și vom parcurge aceeași pași ca și la linia anterioară.

Vom continua până în momentul în care valoarea liniei curente va fi mai mare decât numărul de linii ale matricei.

Pe fiecare linie vom modifica cel mult $\log_2 N$ elemente; în plus, vom modifica elemente de pe cel mult $\log_2 M$ linii, așadar, ordinul de complexitate al operației de modificare a unui element este $O(\log M \cdot \log N)$.

Ca urmare, dacă se modifică elementul de coordonate (3, 1), atunci vor trebui modificate următoarele elemente

ale matricei care reprezintă arborele de arbori indexați binar: c_{31}, c_{32}, c_{34}, c_{41}, c_{42}, c_{44}, c_{81}, c_{82} și c_{84}.

În cazul în care dorim să efectuăm o interogare vom folosi aceeași metodă. Singura diferență este, din nou, faptul că valorile de forma 2^k sunt scăzute în loc să fie adunate.

Datorită faptului că, de data aceasta, avem nevoie de patru sume pentru a furniza rezultatul, algoritmul va fi aplicat de patru ori. Ordinul de complexitate al algoritmului va fi $O(\log M \cdot \log N)$.

Vom prezenta în continuare versiunea în pseudocod a algoritmului de rezolvare a problemei în cazul bidimensional, folosind un arbore de arbori indexați binar.

```
//inițializări
scrie Introduceți dimensiunile matricei:
citește M, N
pentru i ← 0, M execută
  pentru j ← 0, N execută
    c_ij ← 0        //valorile inițiale sunt nule
  sfârșit pentru
sfârșit pentru
scrie Introduceți codul operației
citește cod
cât timp cod ≠ 3 execută
  dacă cod = 1  //modificare
  atunci
    scrie Introduceți indicii elementului care va fi modificat:
      citește indx, indy
    scrie Introduceți valoarea care va fi adunată (valoare negativă pentru scăderi):
      citește val
      pozi ← 0
      cât timp indx ≤ M execută
        pozj ← 0
        j ← indy
        cât timp j ≤ N execută
          c_indx,j ← c_indx,j + val
          cât timp indx & 2^pozj = 0 execută
            pozj ← pozj + 1
        sfârșit cât timp
          j ← j + 2^pozj
          pozj ← pozj + 1
      sfârșit cât timp
        cât timp indx & 2^pozi = 0 execută
          pozi ← pozi + 1
      sfârșit cât timp
        indx ← indx + 2^pozi
        pozi ← pozi + 1
    sfârșit cât timp
    altfel  //interogare
    scrie Introduceți coordonatele colțurilor:
      citește st, sus, dr, jos
      //calcularea primei sume
      s ← 0
      x ← jos
      y ← dr
      pozi ← 0
      cât timp x ≥ 0 execută
        pozj ← 0
        j ← y
        cât timp y ≥ 0 execută
          s1 ← s1 + c_x,j
          cât timp x & 2^pozj = 0 execută
            pozj ← pozj + 1
          sfârșit cât timp
          j ← j - 2^pozj
          pozj ← pozj + 1
        sfârșit cât timp
        cât timp x & 2^pozi = 0 execută
          pozi ← pozi + 1
        sfârșit cât timp
        x ← x - 2^pozi
        pozi ← pozi + 1
      sfârșit cât timp
      s1 ← s
      s ← 0
      //calcularea celei de-a doua sume
      x ← jos
      y ← st - 1
      ...  //se aplică exact aceeași secvență
      s2 ← s
      s ← 0
      //calcularea celei de-a treia sume
      x ← sus - 1
      y ← dr
      ...  //se aplică din nou exact aceeași secvență
      s3 ← s
      s ← 0
      //calcularea celei de-a patra sume
      x ← sus - 1
      y ← dr - 1
      ...  //se aplică din nou exact aceeași secvență
      s4 ← s
      scrie Suma elementelor dreptunghiului este s1
        - s2 - s3 + s4
  sfârșit dacă
  scrie Introduceți codul operației:
  citește cod
sfârșit cât timp
```

Cazul tridimensional

Problema enunțată poate fi modificată astfel încât să lucrăm în spațiul tridimensional. Enunțul ar putea fi următorul:

Se consideră un tablou tridimensional care conține numere întregi și toate elementele sale sunt nule. O modificare a unui element al tabloului constă în adunarea unei valori la valoarea curentă (pot fi realizate și scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unui subtablou "paralelipipedic".

Așadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă paralelipipedică" a tabloului. Evident, pentru reprezentarea datelor vom folosi tablouri tridimensionale.

Și acest enunț poate fi reformulat în diferite moduri. O variantă ar putea fi:

O zonă a spațiului este reprezentată de un paralelipiped format din cuburi cu latura egală cu unitatea. Dimensiunile paralelipipedului sunt M, N și P. În fiecare sector (cub) pot sosi sau pleca nave spațiale. De asemenea, în fiecare moment amiralul flotei poate cere informații referitoare la numărul total de nave spațiale dintr-o regiune paralelipipedică din această zonă a spațiului.

Nu vom mai prezenta pe larg cele trei tipuri de algoritmi; vom face doar câteva precizări care ne vor ajuta să generalizăm problema pentru spații n-dimensionale.

În primul rând, în fiecare poziție în care algoritmii pentru cazul bidimensional conțineau două cicluri imbricate, pentru algoritmii care rezolvă cazul tridimensional vom avea trei cicluri. Ca urmare, ordinul de complexitate al primilor doi algoritmi va deveni $O(M \cdot N \cdot P)$. Ordinul de complexitate al celui de-al treilea algoritm va fi $O(\log M \cdot \log N \cdot \log P)$.

În al doilea rând, elementele tabloului tridimensional vor fi identificate prin trei indici. Așadar, pentru o modificare vom avea nevoie de trei parametri "geometrici", iar pentru o interogare de șase astfel de parametri.

În al treilea rând, pentru ultimii doi algoritmi va trebui să calculăm opt sume în loc de patru. Formula de calcul va fi următoarea:

$$\begin{aligned}\text{Sumă}(x_1, y_1, z_1, x_2, y_2, z_2) = {} & \text{Sumă}(1, 1, 1, x_2, y_2, z_2) - \\ & \text{Sumă}(1, 1, 1, x_1 - 1, y_2, z_2) - \\ & \text{Sumă}(1, 1, 1, x_2, y_1 - 1, z_2) - \\ & \text{Sumă}(1, 1, 1, x_2, y_2, z_1 - 1) + \\ & \text{Sumă}(1, 1, 1, x_1 - 1, y_1 - 1, z_2) + \\ & \text{Sumă}(1, 1, 1, x_1 - 1, y_2, z_1 - 1) + \\ & \text{Sumă}(1, 1, 1, x_2, y_1 - 1, z_1 - 1) - \\ & \text{Sumă}(1, 1, 1, x_1 - 1, y_1 - 1, z_1 - 1)\end{aligned}$$

Mai precizăm faptul că pentru cel de-al treilea algoritm vom folosi un tablou tridimensional care va reprezenta un *arbore de arbori de arbori indexați binar*.

Toate celelalte aspecte ale celor trei algoritmi descriși pot fi adaptate foarte simplu pentru a rezolva problema tridimensională.

Cazul n-dimensional

Vom generaliza acum problema pentru a putea fi enunțată într-un spațiu cu un număr oricât de mare de dimensiuni. O variantă a enunțului este:

Se consideră un tablou n-dimensional care conține numere întregi și toate elementele sale sunt nule. O modificare a unui element al tabloului constă în adunarea unei valori la valoarea curentă (pot fi realizate și scăderi care au forma adunării unor valori negative). Pe parcursul modificărilor pot apărea interogări referitoare la suma elementelor unui subtablou.

Așadar, în acest caz, interogările se referă la suma valorilor dintr-o "zonă n-dimensională" a tabloului. Evident, pentru reprezentarea datelor vom folosi tablouri n-dimensionale.

De data această, cele două cicluri imbricate din algoritmii pentru cazul bidimensional (cele trei din algoritmii pentru cazul tridimensional) vor fi înlocuite de n cicluri imbricate.

Ca urmare, ordinul de complexitate al primilor doi algoritmi va depinde de produsul celor n dimensiuni, iar ordinul de complexitate al celui de-al treilea algoritm va depinde de produsul logaritmilor celor n dimensiuni.

Evident, vom avea acum n parametri "geometrici" pentru o modificare și $2 \cdot n$ parametri de acest tip pentru o interogare. Numărul sumelor care trebuie calculate pentru o interogare (la al doilea și al treilea algoritm) este, pentru cazul general, 2^n.

Ținând cont de aceste observații, algoritmii prezentați pot fi folosiți pentru rezolvarea unor probleme de acest tip în care numărul dimensiunilor este oricât de mare.

Concluzii

În cadrul acestui articol am prezentat o structură de date care permite rezolvarea eficientă a unei categorii de probleme. Pentru a evidenția performanțele algoritmilor care folosesc o astfel de structură de date, am prezentat și alte două categorii de algoritmi, mult mai ușor de implementat, dar ineficienți.

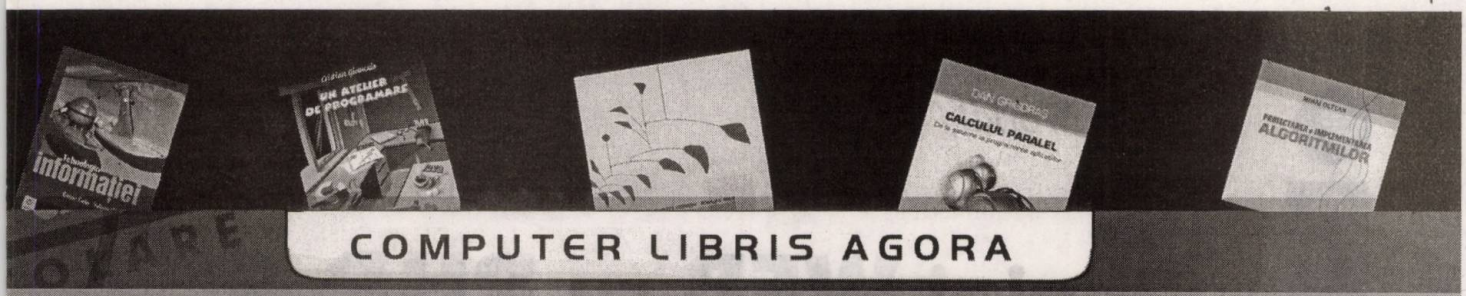
De asemenea, am prezentat modul în care pot fi generalizați algoritmii folosiți pentru cazul cel mai simplu (cel unidimensional) pentru a rezolva problema pentru cazuri în care numărul dimensiunilor este oricât de mare.

Am exemplificat algoritmii pentru cazul în care interogările se referă la suma anumitor elemente. Cu toate acestea este evident faptul că, în cazul tuturor acestor algoritmi, această operație poate fi înlocuită cu altele (produsul elementelor, minimul sau maximul lor etc.).

Bibliografie

1. **J. Nummenmaa, E. Mäkinen, I. Aho,** *IOI '01 Competition*, Tampere, Finland, 2001
2. **P. M. Fenwick,** *A new data structure for cumulative frequency tables*, Software-Practice and Experience, vol. 24, no. 3, p. 327-336, 1994
3. **T. H. Cormen, C. E. Leiserson, R. R. Rivest,** *Introducere în algoritmi*, Computer Libris Agora, Cluj-Napoca, 2000

Mihai Scorțaru este redactor-șef al GInfo. Poate fi contactat prin e-mail la adresa **skortzy@yahoo.com**.



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

În cadrul acestui articol vom începe să prezentăm modul în care putem beneficia de avantajele oferite de PHP în crearea paginilor Web. Vom începe cu descrierea modalității de inserare a codului PHP în documentele folosite pentru generarea paginilor Web și vom descrie apoi modul de utilizare al variabilelor în PHP.

Inserarea codului PHP

Interpretorul *PHP* parcurge documentul accesat până în momentul în care întâlnește un marcaj de deschidere care indică faptul că textul care urmează trebuie interpretat ca fiind cod *PHP*.

În continuare, textul este interpretat ca fiind cod *PHP* până în momentul în care este întâlnit marcajul de închidere.

Întreg textul care nu este interpretat ca fiind cod *PHP* este furnizat la ieșire în forma în care a fost primit ca intrare de către interpretor.

Există mai multe marcaje care indică începerea unei secvențe de cod *PHP*, dar doar două dintre ele sunt folosite de obicei.

Dacă dorim ca documentul să respecte specificațiile *XML*, atunci singura posibilitate de inserare a cosului *PHP* este folosirea unei secvențe de tipul:

```
<?php
  //cod PHP
?>
```

Cea de-a doua posibilitate este utilizarea marcajului **<SCRIPT>** într-o manieră asemănătoare celei folosite pentru includerea de *script*-uri *JavaScript*. Sintaxa este următoarea:

```
<SCRIPT language = "php">
  //cod PHP
</SCRIPT>
```

Exsită alte două posibilități, dar acestea sunt folosite rar, în situații particulare.

Este permisă "ieșirea" și "intrarea" în "modul *PHP*" de oricâte ori este necesar.

De exemplu, următoarea secvență este interpretată corect:

```
<?php
  if ($expresie){
    ?>
    <B>Adevărat!</B>
    <?php
  }
  else{
    ?>
    <B>Fals!</B>
  <?php
  }
?>
```

Secvența este corectă deoarece, după întâlnirea unui marcaj de închidere (care indică terminarea unei secvențe de cod *PHP*), întregul text întâlnit până la următorul marcaj de deschidere (care indică începerea unei secvențe de cod *PHP*) este furnizat la ieșire nemodificat.

Secvența de mai sus ar putea părea ciudată, dar în unele cazuri (când se lucrează cu texte de dimensiuni mari) această variantă este mai eficientă decât furnizarea textului ca ieșire folosind comenzi de tip **echo**.

Instrucțiuni PHP

Pentru ca instrucțiunile *PHP* să fie interpretate corect, acestea trebuie separate prin caracterul `';'`. Marcajul de închidere care semnalează terminarea secvenței de cod *PHP* inserează, implicit, un caracter `';'` după ultima instrucțiune *PHP*. Așadar, după ultima instrucțiune a fiecărui bloc de instrucțiuni *PHP*, acest caracter poate lipsi.

Comentarii

În *PHP*, comentariile pot fi inserate folosind sintaxele din *C*, *C++* și *shell*-urile *Unix*.

Apariția secvenței de caractere `'//'` sau a caracterului `'#'` indică faptul că restul liniei reprezintă un comentariu. Pentru a insera comentarii pe mai multe linii, acestea tre-

buie delimitate de secvența '/*' la început și de secvența '*/' la sfârșit.

Trebuie observat faptul că, dacă după '//' sau apare '#' un marcaj care indică încheierea secvenței de cod *PHP*, atunci acesta este luat în considerare și textul care urmează nu mai este considerat comentariu.

Variabile în PHP

În *PHP*, o variabilă este reprezentată de semnul '$', urmat de numele variabilei. La fel ca în limbajele *C/C++* sau *Java*, pentru denumirea variabilelor se face distincție între literele mici și literele mari.

Numele unei variabile poate începe cu o liniuță de subliniere ('_') sau o literă. Restul caracterelor care formează numele variabilei pot fi litere, cifre sau liniuțe de subliniere. În *PHP*, sunt considerate litere toate caracterele cuprinse între 'a' și 'z', cele cuprinse între 'A' și 'Z', precum și cele care au codul *ASCII* cuprins între 127 și 255.

Începând cu versiunea 4, au fost introduse *referințele*; astfel, o variabilă poate referi o alta, astfel încât modificarea valorii uneia duce la modificarea automată a valorii celeilalte. O referință la o variabilă este reprezentată de caracterul '&'. Iată un scurt exemplu:

```
<HTML>
  <HEAD>
    <TITLE>
      Referințe
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      $a = "Gazeta de Informatică";
      $b = &$a;
      $b = "GInfo";
      echo $a;
      echo "<BR>";
      echo $b;
    ?>
  </BODY>
</HTML>
```

Figura 1

După cum se poate observa în fereastra din figura 1, modificarea valorii variabilei $b a dus la modificarea automată a valorii variabilei $a.

Variabile predefinite

Există o mulțime de variabile care sunt deja definite și care pot fi utilizate în diferite scopuri. În cele ce urmează vom descrie câteva dintre acestea.

Prin intermediul vectorului $GLOBALS pot fi accesate toate variabilele globale care sunt accesibile *script*-ului *PHP* curent; acest vector este indexat chiar prin numele variabilelor globale.

Vectorul $SERVER conține o serie de variabile ale căror valori sunt setate de *server*-ul *Web*; majoritatea valorilor variabilelor din acest vector depind de mediul de execuție al *script*-ului curent.

Vectorii $_GET și $_POST conțin variabile primite de *script* prin intermediul unor transferuri care folosesc metodele *HTTP* get, respectiv post. De exemplu, prin intermediul acestor vectori pot fi accesate valorile câmpurilor dintr-un formular care a fost completat și transmis folosind una dintre cele două metode.

Vectorul $_COOKIE conține valorile variabilelor care conțin informații referitoare la *cookie*-urile păstrate pe calculatorul utilizatorului care accesează pagina *Web*.

Vectorul $_FILES conține variabile primite de *script* prin intermediul încărcărilor de fișiere prin metoda post.

Vectorul $_ENV conține variabile disponibile prin intermediul mediului în care este executat.

Vectorul $_REQUEST conține variabile disponibile prin intermediul oricărui tip de mecanism cu ajutorul căruia utilizatorul poate introduce date.

Vectorul $_SESSION conține variabilele care corespund sesiunii curente a *script*-ului.

Am dat doar definițiile acestor vectori, chiar dacă acestea nu oferă informații suficiente pentru a înțelege modul lor de utilizare. Pe măsură ce vom avea nevoie de astfel de informații vom prezenta detalii referitoare la modul de utilizare.

Domeniul de vizibilitate al variabilelor

În *PHP* o variabilă poate fi accesată doar în contextul în care a fost definită. De exemplu, o variabilă definită în interiorul unei funcții nu va putea fi accesată decât de instrucțiunile din interiorul acelei funcții. Mai mult, în mod implicit, o variabilă definită în exteriorul unei funcții nu va putea fi accesată în interiorul funcției chiar dacă ea nu este redefinită în cadrul funcției.

Pentru ca o funcție să aibă acces la o variabilă definită în afara ei, variabila trebuie redeclarată ca variabilă globală în interiorul funcției.

Să considerăm următorul exemplu:

```
<?php
  $a = 1;
 function Test(){
    echo $a;
  }
  Test();
?>
```

Nu se va afișa valoarea 1 deoarece instrucțiunea **echo** $a se referă la variabila locală $a care nu este definită, deci nu are nici o valoare.

Vom prezenta acum o versiune în care instrucțiunea **echo** $a va accesa variabila $a definită în afara funcției Test():

```
<?php
  $a = 1;
```

```
function Test(){
    global $a;
    echo $a;
}
Test();
?>
```

O a doua posibilitate de a accesa o variabilă definită în afara funcției este folosirea vectorului $GLOBALS:

```
<?php
 $a = 1;
function Test(){
    echo $GLOBALS["a"];
}
Test();
?>
```

O altă facilitate oferită de *PHP* este folosirea variabilelor *statice*. Acestea își păstrează valoarea dacă se iese din contextul în care a fost definită variabila și, ulterior, se revine în acel context. De exemplu, o variabilă statică definită în interiorul unei funcții își păstrează valoarea de la un apel la altul al funcției. Vom arăta acum modul în care pot fi utilizate variabilele statice:

```
<HTML>
  <HEAD>
    <TITLE>
      Variabile statice
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function Test(){
        static $a = 0;
        echo $a;
        echo "<BR>";
        $a = $a + 1;
      }
      Test();
      Test();
      Test();
    ?>
  </BODY>
</HTML>
```

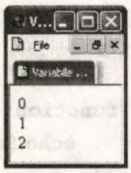


Figura 2

După cum se poate observa în figura 2, la primul apel al funcției `Test()` variabila `$a` are valoarea `0`, la al doilea apel are valoarea `1`, iar la al treilea are valoarea `2`.

Variabile cu nume variabil

Uneori este convenabil să folosim variabile al căror nume poate fi modificat în mod dinamic. Printr-o secvență de tipul `<?php $a = "Hello"; ?>` am creat o variabilă `$a` a cărei valoare este `Hello`. Acum, prin intermediul unei secvențe de tipul `<?php $$a = "World!"; ?>` se creează o variabilă `$Hello` a cărei valoare este `World!`.

În acest moment secvențele `<?php echo "$a ${$a}"; ?>` și `<?php echo "$a $Hello"; ?>` au exact același efect. Codul complet este prezentat în continuare, iar efectul este ilustrat în figura 3.

```
<HTML>
  <HEAD>
    <TITLE>Variabile cu nume variabil</TITLE>
  </HEAD>
  <BODY>
    <?php
      $a = "Hello";
      $$a = "World!";
      echo "$a ${$a}";
      echo "<BR>";
      echo "$a $Hello";
    ?>
  </BODY>
</HTML>
```

Figura 3

Variabile corespunzătoare formularelor

Dacă utilizatorul completează un formular, iar "*feedback*"-ul trebuie furnizat de un *script PHP*, atunci informațiile din formular devin accesibile acestui *script*.

Pentru a indica faptul că datele din formular trebuie procesate de un anumit script *PHP*, atunci numele fișierului care conține *script*-ul trebuie furnizat ca valoarea pentru atributul **action** al marcajului **<FORM>**. Presupunem că avem un formular în care trebuie introdus un nume de utilizator și o parolă, iar aceste date trebuie manipulate de *script*-ul *PHP* stocat în fișierul **form.php**. În aceste condiții, documentul *HTML* care afișează formularul este asemănător cu cel prezentat în continuare (efectul poate fi vizualizat în figura 4):

```
<HTML>
  <HEAD>
    <TITLE>Un formular simplu</TITLE>
  </HEAD>
  <BODY>
    <FORM action = 'form.php' method = post>
      <DIV align = center>
        Numele: 
        <INPUT type = text name = nume> <BR>
        Parola:   
        <INPUT type = password name = parola>
        <BR>
        <INPUT type = submit
               name = submit
          value = "Trimite!">
      </DIV>
    </FORM>
  </BODY>
</HTML>
```

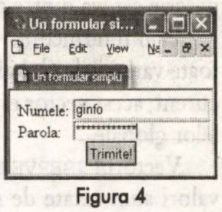


Figura 4

Vom prezenta acum un script care prelucrează datele din formular. Informațiile completate pot fi accesate prin intermediul vectorilor $_POST sau $_REQUEST. Pagina afișată după apăsarea butonului *Trimite!* este prezentată în figura 5.

```
<HTML>
  <HEAD>
    <TITLE>Prelucrare formular</TITLE>
  </HEAD>
  <BODY>
    <?php
      echo "Numele este <B><I> ";
      echo $_POST['nume'];
      echo "</I></B>.<BR>Parola este <B><I> ";
      echo $_REQUEST['parola'];
      echo "</I></B>.<BR>";
    ?>
  </BODY>
</HTML>
```

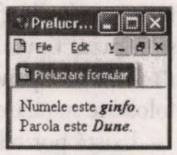


Figura 5

Există posibilitatea de a afișa atât formularul, cât și rezultatul prelucrării datelor, folosind același *script PHP*. Pentru aceasta, va trebui să verificăm la început "starea curentă a *script*-ului". Dacă formularul a fost completat vom afișa rezultatele prelucrării, în caz contrar vom afișa formularul propriu-zis pentru a permite introducerea datelor.

O posibilitate de determinare a stării curente este includerea unui câmp ascuns în formular, căruia i se atribuie o anumită valoare în momentul în care este completat formularul. Așadar, la începutul *script*-ului vom verifica valoarea acestui câmp ascuns.

Vom crea un singur fișier (form.php), iar atributul **action** al marcajului <**FORM**> va avea tot valoarea form.php:

```
<HTML>
  <?php
    if ($_POST['actiune'] == 'trimis'){
  ?>
  <HEAD>
    <TITLE>Prelucrare formular</TITLE>
  </HEAD>
  <BODY>
    <?php
      echo "Numele este <B><I> ";
      echo $_POST['nume'];
      echo "</I></B>.<BR> Parola este <B><I> ";
      echo $_REQUEST['parola'];
      echo "</I></B>.<BR>";
    ?>
  </BODY>
  <?php
    }
    else{
  ?>
  <HEAD>
    <TITLE>Un formular simplu</TITLE>
  </HEAD>
  <BODY>
    <FORM action = 'form.php' method = post>
      <DIV align = center>
        Numele: 
        <INPUT type = text name = nume><BR>
        Parola:   
        <INPUT type = password name = parola>
        <BR>
        <INPUT type = hidden name = actiune
                                value = trimis>
        <INPUT type = submit name = submit
                              value = "Trimite!">
      </DIV>
    </FORM>
  </BODY>
  <?php
  }
  ?>
</HTML>
```

După prima accesare a documentului **form.php** va fi afișată pagina din figura 4. După apăsarea butonului *Trimite!* va fi accesat același document, dar va fi afișată pagina din figura 5.

În cazul în care butonul prin intermediul căruia sunt trimise informațiile completate este, de fapt, o imagine (inserată printr-o secvență de tipul <**INPUT type** = "image" **src** = "imagine.gif" **name** = "sub">) atunci, pe lângă variabilele corespunzătoare câmpurilor, *script*-ul *PHP* va putea utiliza alte două variabile (sub_x și sub_y) ale căror valori reprezintă coordonatele punctului din imagine, pe care se afla cursorul *mouse*-ului în momentul în care s-a efectuat *click*-ul prin intermediul căruia a fost comunicată programului de navigare intenția de transferare a informațiilor completate.

Constante în PHP

La fel ca în majoritatea limbajelor de programare, constantele sunt identificatori care sunt folosiți pentru accesarea unei valori care nu se poate modifica. Denumirea constantelor respectă aceleași reguli ca și denumirea variabilelor, dar numele lor nu trebuie precedate de semnul '$'.

În *PHP*, constantele pot fi definite folosind funcția define(). Nu este posibilă definirea de constante prin intermediul unor atribuiri simple. Pentru a defini o constantă a cărei valoare este Hello World!, putem folosi secvența: <?php define("MESAJ, "Hello World!"); ?>.

Așadar, valoarea constantei MESAJ va fi întotdeauna Hello World!.

Există o mulțime de constante predefinite care pot fi accesate de *script*-urile *PHP*. Acestea se împart în două categorii: constante standard și constante specifice nucleului *PHP* folosit.

Vom prezenta valorile acestor constante în momentul în care le vom folosi.

În *PHP* pot fi definite doar constante care reprezintă valori logice, numere întregi, numere reale sau șiruri de caractere.

Tipuri de date în PHP

Există opt tipuri elementare, grupate în trei categorii. Patru dintre acestea sunt tipuri scalare (`boolean`, `integer`, `float` și `string`), două sunt tipuri compuse (`array` și `object`), iar alte două sunt tipuri speciale (`resource` și `NULL`).

De asemenea, din motive de lizibilitate, au fost introduse trei pseudotipuri: `mixed`, `number` și `callback`. Mai există și tipul `double`, dar semnificația acestuia este aceeași cu cea a tipului `float`. Cele două denumiri coexistă doar din motive "istorice".

În *PHP*, de obicei, tipul unei variabile nu este specificat de către programator, ci este stabilit în timpul execuției în funcție de contextul în care este folosită variabila.

Tipul boolean

Variabilele de acest tip pot avea doar două valori: `ADEVĂRAT` sau `FALS`. Aceste valori pot fi indicate prin cuvintele cheie **`TRUE`** sau **`FALSE`** (pentru ambele nu se face distincție între literele mici și literele mari).

Există posibilitatea de a converti o variabilă de orice tip la tipul `boolean`. În momentul efectuării unei conversii, sunt convertite la valoarea **`FALSE`** următoarele valori:

- numărul întreg `0`;
- numărul real `0.0`;
- șirul vid;
- șirul `"0"`;
- un vector fără nici un element;
- un obiect fără nici o variabilă membru;
- o variabilă de tipul `NULL`;
- o variabilă nedefinită.

Orice altă valoare este convertită la valoarea **`TRUE`** (inclusiv resursele).

Tipul integer

O variabilă de tip `integer` reprezintă o valoare din mulțimea numerelor întregi. Aceste numere pot fi specificate în baza 10, în baza 16 sau în baza 8, convențiile fiind aceleași ca și în limbajele *C/C++* sau *Java*.

Modul de reprezentare depinde de platforma utilizată; de obicei se folosește reprezentarea pe 32 de biți.

Interpretorul *PHP* nu oferă suport pentru numere întregi fără semn.

Trebuie remarcat faptul că în *PHP* nu există nici un operator pentru efectuarea de împărțiri întregi. De exemplu, rezultatul operației `3 / 2` nu va fi un numărul întreg `1` (ca în *C/C++* sau *Java*), ci numărul real (`float`) `1.5`.

Și pentru numerele întregi există posibilitatea efectuării de conversii:

- valoarea logică **`TRUE`** este convertită la valoarea întreagă `1`;
- valoarea logică **`FALSE`** este convertită la valoarea întreagă `0`;
- un număr real este convertit prin "rotunjire înspre `0`"; așadar, valoarea reală `2.5` va fi convertită la valoarea întreagă `2`, în timp ce valoarea reală `-2.5` va fi convertită la valoarea întreagă `-2`;
- un șir de caractere este convertit luând în considerare doar primele caractere care conțin informații numerice; așadar șirul `"10"` va fi convertit la valoarea întreagă `10`; de asemenea șirul `"10 ani"` va fi convertit tot la valoarea `10`; dacă primele caractere nu conțin informații numerice, rezultatul conversiei va fi valoarea `0`.

Tipul float

O variabilă de tip `float` poate fi specificată folosind fie forma zecimală, fie cea științifică (cu exponent). La fel ca și în cazul tipului `integer`, precizia variabilelor de tipul `float` este dependentă de platforma utilizată. De obicei se folosește standardul *IEEE* 64.

Există posibilitatea de a converti o variabilă de orice tip la tipul `float`. Pentru numerele reale se pot efectua următoarele conversii:

- un șir de caractere este convertit luând în considerare doar primele caractere care conțin informații numerice; așadar șirul `"10.2"` va fi convertit la valoarea reală `10.2`; șirul `"1.23E1 ani"` va fi convertit la valoarea `12.3`;
- în toate celelalte cazuri se realizează conversii la numere întregi care apoi sunt convertite la valorile reale corespunzătoare.

Tipul string

O variabilă de tip `string` reprezintă un șir de caractere. Un caracter se reprezintă pe un octet, deci sunt 256 de caractere distincte. Acest lucru implică faptul că interpretorul *PHP* nu oferă suport nativ pentru setul de caractere *Unicode*. Lungimea variabilelor de tip `string` nu este limitată de către interpretor.

Literalii de tip șir de caractere pot fi specificați în trei moduri diferite:

- prin folosirea ghilimelelor simple (exemplu `$a = 'acesta este un șir de caractere'`). Pentru a putea avea în cadrul șirului de caractere simbolul `"'"`, atunci înaintea acestuia trebuie scris caracterul `"\"`, iar pentru a putea specifica simbolul `"\"` acesta trebuie dublat. De exemplu, rezultatul instrucțiunii **`echo`** `'acesta este un șir de caractere în care apar caracterele "\\" și "\'"'` este: `acesta este un șir de caractere în care apar caracterele "\" și "'"`;
- prin folosirea ghilimelelor duble. Folosind această notație pot fi specificate mai multe caractere speciale, pe lângă caracterele de la varianta anterioară, printre care: sfârșit de linie (`"\r"`), rând nou (`"\n"`), tab orizontal (`"\t"`), semnul dolar (`"\$"`), ghilimele duble (`"\""`), secvențe de caractere pentru specificarea faptului că o expresie regulară este în notație octală (`"\[0-7]{1,3}"`) și secvențe de caractere pentru specificarea faptului că o expresie

regulară este în notație hexazecimală("\x[0-9A-Fa-f]{1,2}"). Cel mai important lucru este acela că, folosind acest mod de specificare a literalilor de acest tip, numele de variabile care apar în interior vor fi transformate în valoarea lor. De exemplu, dacă $a este o variabilă de tipul integer și are valoarea 2, atunci șirul de caractere "Variabila a are valoarea $a." va fi transformat în șirul "Variabila a are valoarea 2.".

- notația ***heredoc***. Acest tip de notație a fost introdus la versiunea 4 a interpretorului *PHP*. Pentru a specifica un șir de caractere folosind această notație trebuie utilizat operatorul "<<<" urmat de un identificator ales de utilizator. Toate caracterele care se află între operatorul "<<<" urmat de un identificator pe o singură linie, și același identificator pe o altă linie vor constitui valoarea șirului de caractere. De exemplu, instrucțiunea:

```
$str = <<<SF
Acesta este un exemplu
de utilizare a sintaxei
heredoc
SF;
```

va avea ca rezultat un șir de caractere format din trei linii de text.

Pentru a accesa un anumit caracter din șirul de caractere se folosește, după numele variabilei de tip string, indicile caracterului care trebuie accesat scris între acolade. De exemplu, $str{0} returnează primul caracter din șirul de caractere $str.

În cazul în care dorim să concatenăm două șiruri de caractere vom folosi operatorul ".". Folosirea operatorului "+" nu va concatena cele două șiruri.

Există posibilitatea de a converti o variabilă de orice tip la tipul string. Pentru șirurile de caractere se pot efectua următoarele conversii:

- valoarea logică **TRUE** va fi convertită la șirul "1", iar valoarea logică **FALSE** va fi convertită la șirul vid ("");
- un număr întreg va fi convertit la un șir de caractere care reprezintă valoarea numărului în baza 10;
- un număr real va fi convertit la un șir de caractere care reprezintă notația științifică a acestuia;
- obiectele sunt întotdeauna convertite la șirul "Object";
- variabilele de tipul resource sunt convertite la șirul "Resource id #n", unde n reprezintă un număr unic atașat reursei respective de către interpretorul *PHP*;
- valoarea **NULL** este convertită la șirul vid ("").

Tipul array

Vectorii în *PHP* sunt niște mulțimi formate din *chei*. Fiecărei chei din vector i se atașează o valoare. Acest tip de date este optimizat astfel încât să poată fi folosit în locul următoarelor structuri de date: liste, tabele de dispersie, dicționare, colecții, stive, cozi și altele. Datorită faptului că o valoare poate fi reprezentată de un alt vector, se pot simula foarte ușor arborii *n*-dimensionali sau tablourile *n*-dimensionale.

Valoarea unei variabile de tip vector se poate specifica folosind construcția array(cheie => valoare, cheie => valoare, ...).

De exemplu, următoarea instrucțiune *PHP* va construi un vector cu două elemente, dintre care unul este de tip string, iar celălat de tip boolean:

```
$a = array("ch" => "string", 12 => TRUE);
```

Variabila $a reprezintă un vector; $a["ch"] are valoarea string, iar $a[12] are valoarea **TRUE**.

În cazul în care nu se specifică o cheie pentru o valoare, atunci acea valoare va fi atașată unei chei care va fi cheia maximă de tip integer folosită anterior, la care se adaugă valoarea 1. Cheile pot avea și valori negative. Dacă nu există chei de tip integer, atunci valoarea va fi atașată cheii 0. De exemplu, următoarele două instrucțiuni sunt echivalente:

```
array(5 => 43, 32, 56, "b" => 12);
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
```

Dacă se folosește valoarea logică **TRUE** ca și cheie, atunci aceasta va fi convertită la cheia de tip întreg 1, iar valoarea **FALSE** va fi convertită la numărul întreg 0.

Nu se pot folosi pentru chei variabile de tipul array sau object.

O variabilă de tip array se poate modifica prin setarea explicită de valori. De exemplu instrucțiunea $a["x"] = 42; adaugă în vectorul $a valoarea 42 atașată cheii "x".

Dacă se folosește un vector care nu a fost definit anterior, atunci acesta este creat automat. Așadar, printr-o instrucțiune de forma $a[5] = 42, în cazul în care vectorul $a nu există, atunci se va crea un vector cu un singur element. Cheia acestuia va fi numărul întreg 5, iar valoarea sa va fi 42.

De asemenea, există posibilitatea de a crea un element nou fără a-i preciza cheia. Sintaxa are forma $vector[] = valoare; această instrucțiune are ca efect adăugarea unui element a cărui cheie este un număr întreg mai mare cu 1 decât cel mai mare număr întreg care este cheie a unui alt element al vectorului. Dacă nu există nici o astfel de cheie, atunci noul element va avea cheia 0. De exemplu, următoarele două secvențe sunt echivalente:

```
$a[5] = 1;          $a[5] = 1;
$a[6] = 2;          $a[] = 2;
```

Prin conversia la un vector a unei variabile de tip scalar (boolean, integer, float, string) sau resource se creează un vector cu un singur element; cheia acestui element este numărul întreg 0, iar valoarea este cea a variabilei convertite.

Dacă se convertește un obiect (variabilă de tip object), atunci vectorul rezultat va conține câte un element pentru fiecare variabilă membru a obiectului. Cheile elementelor vor fi date de denumirile proprietăților obiectului (variabilele membru ale obiectului), iar valorile elementelor vor fi valorile proprietăților obiectului.

Dacă realizăm o conversie a unei variabile de tip NULL, atunci rezultatul va fi un vector vid (care nu conține nici un element).

Vom prezenta în continuare câteva exemple în care vom descrie mai detaliat posibilitățile oferite de folosirea vectorilor în *PHP*.

Pentru început, vom prezenta un vector ale cărui elemente reprezintă caracteristicile unei portocale. Veți observa că există mari deosebiri dintre vectorii din *PHP* și cei din *Pascal* sau *C/C++*. De fapt, acest vector este echivalent cu tipul înregistrare din aceste limbaje. Așadar, vectorul care conține caracteristicile portocalei poate fi definit astfel:

```
$a = array('denumire' => 'portocala',
           'familie' => 'citrice',
           'culoare' => 'portocaliu',
           'forma' => 'rotunda',
           'gust' => 'dulce'
          );
```

Putem adăuga și alte elemente, care să reprezinte diferite alte proprietăți. De exemplu, am putea avea nevoie de o valoare suplimentară căreia nu dorim să îi atribuim nici un nume de identificare (cheie). Pentru ca vectorul să conțină un element suplimentar cu valoarea 4, vom putea defini vectorul astfel:

```
$v = array('denumire' => 'portocala',
           'familie' => 'citrice',
           'culoare' => 'portocaliu',
           'forma' => 'rotunda',
           'gust' => 'dulce',
           4
          );
```

Cheia elementului cu valoarea 4 va fi numărul întreg 0 deoarece nu există nici o altă cheie care este număr întreg. O alternativă de construire a acestui vector este următoarea:

```
$v['denumire'] = 'portocala';
$v['familie' = 'citrice';
$v['culoare' = 'portocaliu';
$v['forma' = 'rotunda';
$v['gust' = 'dulce';
$v[] = 4;
```

Putem construi și vectori similari celor din *Pascal* sau *C/C++*.

De exemplu, un vector care conține primele cinci numere naturale strict pozitive și are indicii cuprinși între 0 și 4 poate fi construit astfel.

```
$v[] = 1;
$v[] = 2;
$v[] = 3;
$v[] = 4;
$v[] = 5;
```

Construcțiile echivalente din *Pascal* și *C/C++* sunt următoarele:

Varianta Pascal:

```
const v:array[0..4] of Longint = (1, 2, 3, 4, 5);
```

Varianta C/C++:

```
long v[5] = {1, 2, 3, 4, 5};
```

În *PHP* există și alte modalități de creare a aceluiași vector. Două dintre acestea sunt: `$v = array(0 => 1, 1 => 2, 2 => 3, 3 => 4, 4 => 5);` și `$v = array(1, 2, 3, 4, 5);`.

Evident, există posibilitatea de a crea vectori care au ca elemente valori care nu respectă nici o regulă aparentă. Iată un exemplu:

```
$v = array(10,          // cheia este 0
           5 => 6,      // cheia este 5
           3 => 7,      // cheia este 3
           'a' => 4,    // cheia este 'a'
           11,          // cheia este 6
           '8'  =>  2,  // cheia este 8 (întreg!)
           '02' => 77,  // cheia este '02'
           0 => 12      // cheia este 0; valoarea 12
                        // suprascrie valoarea 10
          );
```

Putem crea și vectori vizi (care nu conțin nici un element); pentru aceasta vom folosi o instrucțiune de tipul: `$vector_vid = array();`.

Elementele unui vector pot fi considerate ca reprezentând o colecție de date.

Următorul exemplu ilustrează modul de utilizare a unui astfel de vector (pagina *Web* generată este prezentată în figura 6):

```
<HTML>
  <HEAD>
    <TITLE>Colection</TITLE>
  </HEAD>
  <BODY>
    <?php
      $colors = array('red',
                      'green',
                      'blue',
                      'silver'
                     );
      foreach ($colors as $color){
        echo "<FONT size = 4 color = $color>";
        echo "<B>This text is <I>$color!</I>";
        echo "</B></FONT><BR>";
      }
    ?>
  </BODY>
</HTML>
```

Figura 6

Nu este posibilă modificarea valorilor unui vector folosind un astfel de ciclu. O posibilitate de a rezolva această problemă este folosirea unei secvențe de tipul celei prezentate în exemplul următor.

```
<HTML>
  <HEAD>
    <TITLE>Modificare</TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $colors = array('red', 'green', 'blue',
                                        'silver');
      foreach ($colors as $key => $color){
        $colors[$key] = strtoupper($color);
      }
      echo "<TT>";
      print_r($colors);
      echo "</TT>"
    ?>
  </BODY>
</HTML>
```

Pagina *Web* generată este asemănătoare cu cea prezentată în figura 7.

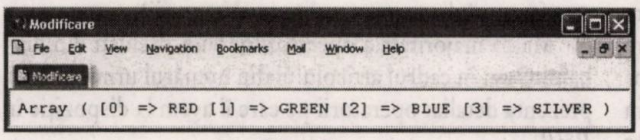


Figura 7

Vom prezenta acum un exemplu în care un vector este completat cu numele fișierelor și subdirectoarelor din directorul curent. Aceste denumiri vor fi afișate într-o pagină *Web* (prezentată în figura 8).

```
<HTML>
  <HEAD>
    <TITLE>Fisiere</TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $handle = opendir('.');
      while ($file =
            readdir($handle)){
        $files[] = $file;
      }
      echo "<TT>";
      foreach ($files as $file){
        echo "$file<BR>";
      }
      closedir($handle);
      echo "</TT>";
    ?>
  </BODY>
</HTML>
```

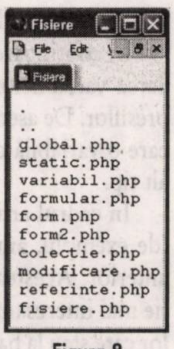


Figura 8

Datorită faptului că valoarea unui element al unui vector poate avea orice tip, elementele pot fi vectori. Așadar, pot fi create tablouri multidimensionale care au orice structură dorită. Un exemplu este prezentat în continuare:

```
fructe = array("fructe" => array("a" => "mere",
                                 "b" => "pere",
                                 "c" => "alune"
                                 ),
               "numere" => array(1, 2, 3, 4, 5, 6),
               "metale" => array("platină",
                                 5 => "aur",
                                 "argint"
                                 )
              );
```

În aceste condiții, de exemplu, comanda **echo** `$fructe["metale"][5];` duce la afișarea șirului `aur`. De asemenea, comanda **echo** `$fructe["fructe"]["b"];` determină afișarea șirului `pere`.

Pot fi create direct și tablouri multidimensionale prin instrucțiuni de tipul `$suc["mere"]["verde"]="gustos";`.

Tipul object

Vom prezenta într-un articol viitor mai multe detalii referitoare la obiectele *PHP*. Acum vom face doar câteva precizări generale referitoare la acest tip de date.

Pentru a defini un obiect care poate fi folosit pentru afișarea mesajului `Hello World!` scrie următoarea secvență:

```
class HelloWorld{
  function DisplayHelloWorld(){
    echo "Hello World!";
  }
}
```

Pentru a utiliza o variabilă de tip obiect va trebui să realizăm o ***instanțiere*** prin intermediul instrucțiunii **new**; sintaxa este următoarea:

```
$a = new HelloWorld;
```

Astfel, variabila `$a` devine un obiect ale cărui metode pot fi utilizate. Pentru afișarea propriu-zisă a mesajului va trebui să executăm metoda `DisplayHelloWorld()` printr-o instrucțiune de tipul `$a->DisplayHelloWorld();`.

Orice variabilă de un anumit tip poate fi convertită într-un obiect. Dacă variabila respectivă este un obiect, atunci ea nu va fi modificată. În caz contrar, efectul conversiei este crearea unei noi instanțe a clasei `stdClass`. Dacă variabila are tipul `NULL`, atunci noua instanță va fi vidă. În toate celelalte cazuri instanța va conține o variabilă membru numită `scalar` a cărei valoare va fi cea a variabilei convertite. Pentru conversii vom folosi instrucțiuni de tipul `$obiect = (object)"Hello World!"`. După realizarea conversiei vom putea tipări mesajul `Hello World!` folosind instrucțiunea **echo** `$obiect->scalar;`.

Tipul resource

Variabilele de tip `resource` sunt folosite pentru păstrarea unor referințe către anumite resurse externe cum ar fi conexiuni la baze de date, fișiere etc. Resursele sunt create și utilizate de anumite funcții speciale. Datorită specificului acestui tip de date valoarea nici unei variabile de alt tip nu poate fi convertită la tipul `resource`.

Tipul NULL

Valoarea specială `NULL` este atribuită oricărei variabile care nu a fost inițializată. Această valoare este singura pe care o pot avea variabilele de tip `NULL`. Se consideră că o variabilă are tipul `NULL` dacă:

- i s-a atribuit constanta `NULL`;
- nu a fost inițializată;
- a fost dezinițializată (prin intermediul funcției `unset()`).

Tipul unei variabile

Interpretorul *PHP* nu necesită (de fapt nici nu permite) definirea explicită a tipului unei variabile. Acest tip este determinat în funcție de contextul în care este utilizată variabila. Așadar, dacă unei anumite variabile i se atribuie un șir de caractere, tipul ei devine `string`; dacă, ulterior, i se atribuie un număr întreg, tipul ei devine `integer`.

Interpretorul *PHP* este capabil să realizeze conversii automate în unele situații. Un exemplu în acest sens este utilizarea operatorului de adunare (+). Dacă unul dintre operanzi are tipul `float`, atunci toți operanzii sunt convertiți în numere reale, iar rezultatul este un număr real. În toate celelalte cazuri operanzii sunt convertiți în numere întregi, rezultatul fiind un număr întreg. Trebuie menționat faptul că tipurile operanzilor ***nu se schimbă***, singura modificare este legată de modul în care sunt interpretate valorile acestora în momentul efectuării adunării.

Dacă dorim ca o anumită variabilă să fie convertită la un anumit tip, atunci putem utiliza operatorii de conversie care vor fi descriși în cadrul acestui articol. Este posibil să modificăm explicit tipul unei variabile prin intermediul funcției `settype()`. Funcția pereche (`gettype()`) returnează tipul pe care îl are o anumită variabilă în momentul apelării acestei funcții.

Realizarea conversiilor

Așa cum am afirmat anterior, pentru a impune considerarea unei variabile ca având un anumit tip vom putea folosi operatorii de conversie. Modul de utilizare este foarte simplu: pentru a realiza conversia unei variabile la un anumit tip, aceasta va fi precedată de operatorul de conversie; între operator și variabilă pot apărea spații. Lista operatorilor de conversie este prezentată în continuare:

- (**`int`**) sau (**`integer`**) - variabila este considerată a fi un număr întreg;
- (**`bool`**) sau (**`boolean`**) - variabila este considerată a fi o valoare logică (**`TRUE`** sau **`FALSE`**);
- (**`float`**), (**`double`**) sau (**`real`**) - variabila este considerată a fi un număr real;
- (**`string`**) - variabila este considerată a fi un șir de caractere;
- (**`array`**) - variabila este considerată a fi un vector;
- (**`object`**) - variabila este considerată a fi un obiect;

Imediat după paranteza deschisă și imediat înaintea parantezei închise este permisă apariția spațiilor. Așadar construcțiile `$a = (int)$b` și `$a = ( int )$b` sunt echivalente.

Expresii în PHP

Cea mai intuitivă definiție a unei expresii este dată de sintagma "*orice are o valoare*". Cele mai simple expresii sunt cele formate din variabile sau constante. De exemplu, constanta `5` este o expresie a cărei valoare este `5`, o variabilă căreia i s-a atribuit valoarea `6` este o expresie a cărei valoare este `6` etc.

O altă categorie de expresii este reprezentată de funcții. Acestea returnează întotdeauna o valoare care este considerată a fi valoarea expresiei corespunzătoare apelului funcției.

Cu ajutorul operatorilor pot fi construite expresii mai complexe. Majoritatea operatorilor sunt binari (noua expresie este formată din două subexpresii și un operator). Există mai mulți operatori unari (expresia este formată din operator și o altă expresie) și un operator ternar (expresia este formată din operator și trei subexpresii).

Marea majoritate a operatorilor sunt preluați din limbajul *C++*; în cadrul articolului din numărul următor vom prezenta detaliat operatorii pe care îi avem la dispoziție în *PHP*.

Concluzii

În cadrul acestui articol am prezentat câteva elemente de bază ale limbajului *PHP*.

Am descris modul în care este inserat codul *PHP* în documentele care duc la generarea de pagini *Web*, precum și sintaxa generală a unei instrucțiuni *PHP*.

În continuare am prezentat detalii referitoare la utilizarea variabilelor, constantelor, tipurilor de date și a expresiilor. De asemenea, au fost prezentate modalitățile prin care o variabilă de un anumit tip poate fi convertită la un alt tip.

În cadrul articolului am folosit unele elemente simple (de exemplu, anumite funcții) care nu au fost prezentate anterior. Acestea au fost utilizate pentru ca exemplele să fie mai interesante și nu împiedică înțelegerea mecanismelor care stau la baza utilizării elementelor prezentate. În numerele următoare vom prezenta detaliat toate aceste elemente.

În următorul artciol vom descrie operatorii care pot fi folosiți în *PHP*, precum și principalele structuri de control puse la dispoziție de acest interpretor.

Mihai Scorțaru este redactor-șef GInfo și poate fi contactat prin e-mail la adresa **skortzy@yahoo.com**. *Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa* **csoroiu@yahoo.com**.

TEOREME de programare

Clara Ionescu

După ce am făcut cunoștință cu "teoremele" numite *Prelucrarea unui șir de date*, *Decizia*, *Selecția* și *Căutarea* (*secvențială*), urmează să învățăm cum se implementează subalgoritmii de *numărare* (contorizare).

Numărarea

Fie următoarele probleme propuse spre rezolvare:

P12. Cunoscând numărul familiilor dintr-un județ, valoarea veniturilor lor, precum și valoarea "coșului minim", să se stabilească numărul familiilor care trăiesc sub venitul minim de trai!

P13. Se consideră rezultatele *Olimpiadei Naționale de Informatică* și punctajul minim necesar pentru a putea participa la barajul pentru a intra în lotul național. Să se calculeze procentul participanților care vor intra la baraj!

P14. Să se numere câte vocale conține un text dat!

Datele de intrare pentru problemele de mai sus sunt șiruri de valori (numere sau caractere). Rezultatul cerut este un număr care reprezintă câte elemente din șir au o anumită proprietate. Enunțurile nu garantează că există cu siguranță un astfel de element, deci soluția furnizată poate fi și 0.

Având în vedere că fiecare element trebuie verificat, deoarece este posibil ca (și) ultimul să aibă proprietatea cerută, vom lucra cu o structură repetitivă de tipul **Pentru**.

Vom prezenta în continuare câteva detalii privind declarațiile din programul care va conține subalgoritmul care rezolvă o asemenea problemă:

```
n: întreg          {numărul elementelor șirului de prelucrat}
x: tablou(1..n): tip element          {elementele șirului dat}
bucăți: întreg                        {rezultatul}
Subalgoritm Numărare(n,x,buc):
  buc := 0
  Pentru i=1,n execută
    Dacă P(X[i]) atunci
    buc := buc + 1
  Sfârșit pentru
Sfârșit subalgoritm
```

În cele ce urmează vom prezenta implementarea subprogramelor care rezolvă problemele P12-P14.

```
{avem definit tipul Tablou = array[1..10000] of Real}
{și tipul Membri = array[1..10000] of Byte, pentru un}
{vector cu numărul membrilor celor n familii}
procedure P12(n:Byte; X:Tablou; nr:Membri;
              min:Real; var buc:Byte);
var i:Byte;                    {min = venitul minim de trai}
begin
  buc:=0;
  for i:=1 to n do
    if X[i]/nr[i] < min then
    Inc(buc)
end;
```

P13. Această problemă cere calcularea unui procent, dar acesta se realizează, bineînțeles, tot pe baza unei numărări. De data aceasta, vom implementa subalgoritmul `Numărare` cu ajutorul unei funcții.

```
function procent(n:Byte; X:Tablou; pct:Byte):
                                              Real;
{avem definit tipul Tablou = array[1..250] of Byte}
var buc, i:Byte;
begin
  buc:=0;{încă nu am găsit nici un elem. cu propr. cerută}
  for i:=1 to n do {pct = punctaj necesar pentru baraj}
    if X[i] >= pct then
    Inc(buc);      {numărul elem. având propr. cerută}
  procent:=(100*buc)/n {calcularea procentului cerut}
end;
```

P14.

```
function nrvoc(text:string; vocale:mult):Byte;
{avem definit tipul mult = set of Char, iar variabila vo-
cale conține mulțimea vocalelor, litere mici și mari}
var buc,i:Byte;
begin
  buc:=0;
  for i:=1 to Length(text) do
    if text[i] in vocale then
    Inc(buc);
  nrvoc:=buc
end;
```

COMPRESIA datelor

Claudiu Soroiu

Termenul de comprimare a datelor a apărut în contextul în care se manifestă o necesitate evidentă de a atinge rate mari de transfer în rețele sau de a stoca o cantitate cât mai mare de informații folosind cât mai puțin spațiu. Compresia datelor este necesară în zilele noastre și este foarte des folosită, mai ales în domeniul aplicațiilor multimedia. De-a lungul câtorva episoade vă vom prezenta cele mai des utilizate metode de compresie a datelor.

Istoria compresiei datelor a început pe la jumătatea secolului al *XX*-lea. În perioada care a urmat, cercetători ca *Claude E. Shannon*, *David Huffman*, *Newton Faller*, *Robert G. Gallager*, *Donald E. Knuth*, *Jeffrey Scott Vitter*, *Michael Burrows*, *David J. Wheeler* etc. au adus contribuții importante acestui subdomeniu al matematicii.

Introducere

Pentru a putea "pătrunde" în interesantul domeniu al compresiilor de date avem nevoie de câteva elemente utile, unele dintre ele fiind preluate din cadrul teoriei statisticii și probabilităților.

În cele ce urmează vom prezenta termenii pe care îi vom utiliza în descrierea mecanismelor necesare pentru comprimarea datelor și pentru analiza performanțelor unui algoritm de compresie.

Noțiuni teoretice

Teoria informației este acel domeniu al matematicii care se ocupă cu transmiterea, stocarea și reprezentarea informațiilor.

Compresia datelor este unul dintre subdomeniile *teoriei informației*.

Folosind un sistem digital, anumite informații, cum ar fi secvențele audio și video, care în realitate sunt secvențe continue, nu se pot reprezenta în sistemului digital la fel ca în lumea reală, fapt pentru care acestea trebuie cuantificate. Prin cuantificarea datelor anumite informații se pierd. Cuantificarea datelor constă în eliminarea sau modificarea unor secvențe de date care nu sunt percepute de către om sau care nu modifică prea mult funcționarea unor aplicații. În cazul unor secvențe de date, prin procesul de cuantificare se păstrează numai cadrele care se află la o distanță *d* unele de altele (distanța *d* nu este constantă, ea poate să varieze de la cadru la cadru în funcție de tipul de cuantificare care se aplică și de proprietățile pe care le au datele care trebuie cuantificate). Cu cât distanța dintre două cadre consecutive este mai mică, cu atât calitatea secvenței este mai bună.

Cuantificarea datelor reprezintă un prim pas în cadrul compresiei de date deoarece se reduce cantitatea de informație care trebuie stocată. Cu cât distanța dintre două cadre consecutive obținute prin cuantificare este mai mare, cu atât cantitatea de informație care trebuie stocată este mai mică. Cuantificarea datelor este necesară numai pentru datele multimedia și are ca dezavantaj faptul că se pierd date.

Un ***eveniment*** reprezintă rezultatul produs în urma efectuării unui experiment. Un *eveniment* se numește ***eveniment elementar*** dacă nu poate fi definit ca fiind o reuniune de două evenimente distincte.

Probabilitatea este o măsură definită pe un câmp (mulțime) de evenimente, $p: \Omega \rightarrow [0, 1]$. Probabilitatea $p(A)$ de realizare a unui mesaj (eveniment) A, determinată experimental, este egală cu raportul dintre numărul de cazuri favorabile evenimentului A (n_A) și numărul total de cazuri (n):

$$p(A) = n_A / n.$$

În cazul în care cunoaștem faptul că două evenimente A și B au apărut simultan de n_{AB} ori, iar evenimetul A a apărut de n_A ori și dorim să aflăm probabilitatea de apariție a evenimentului B dacă apare evenimentul A, atunci această probabilitate este egală cu raportul dintre numărul de apariții ale celor două evenimente și numărul de apariții ale evenimentului A și se notează prin $p(B/A)$:

$$p(B/A) = (n_{AB} / n) / (n_A / n) = p(A, B) / p(A),$$

unde $p(A, B)$ reprezintă probabilitatea de apariția simultană a evenimentelor A și B. Rezultă că, dacă avem două eve-

nimente A și B, atunci $p(A, B) = p(A) \cdot p(B/A) = p(B) \cdot p(A/B)$ (relația lui *Bayes*).

Dacă A_i, $1 \le i \le m$, sunt evenimentele elementare ale unui experiment probabilistic, atunci $\sum_{i=1}^{m} p(A_i) = 1$. Această formulă poartă numele de *formula fundamentală a probabilităților evenimentelor elementare*.

Fie A_i, $1 \le i \le m$, evenimentele elementare ale unui experiment probabilistic și B un eveniment oarecare pentru același experiment. În aceste condiții avem:

$$p(B) = \sum_{i=1}^{m} p(A_i) \cdot p(B / A_i)$$

Se spune despre două evenimente că sunt ***echiprobabile*** dacă au aceeași probabilitate de apariție. De exemplu, evenimentele A = "*la următoarea aruncare a unei monede, aceasta va cădea cu fața în sus*" și B = "*la următoarea aruncare a unei monede, aceasta va cădea cu fața în jos*" au aceeași probabilitate de apariție: $p(A) = p(B) = 0,5 = 50\%$.

O ***sursă de informație*** este un experiment prin care se selectează un eveniment dintre n evenimente posibile, folosind un criteriu aleator.

Cantitatea de informație (h) furnizată de un eveniment A este egală cu: $h(A) = \log_2 (1/p(A)) = -\log_2 p(A)$ biți, unde $p(A)$ este probabilitatea de apariție a evenimentului A. Această mărime a fost introdusă de *Shannon*.

Bitul, pe lângă faptul că reprezintă unitatea atomică (care nu mai poate fi divizată) de măsură a informației, mai poate fi definit ca fiind o variabilă ale cărei valori sunt doar 0 și 1.

În teoria informației evenimentele sunt constituite de apariția unor simboluri (caractere).

O sursă de informație este ***discretă*** dacă între elementele generate de ea nu există continuitate. De exemplu, o funcție este discretă dacă nu are nici un punct de continuitate. De exemplu, funcția următoare este discretă:

$$f : R \to R, f(x) = \begin{cases} 0, x \in Q \\ 1, x \in R/Q \end{cases}.$$

Despre o sursă de informație se spune că este ***fără memorie***, dacă apariția unui simbol nu depinde de nici unul dintre simbolurile care au apărut anterior. În cazul în care apariția unui simbol depinde de apariția ultimelor m simboluri anterioare, atunci despre sursa de informație se spune că este o ***sursă cu memorie (Markov) cu m pași***.

După cum se știe din fizică, ***entropia*** reprezintă o mărime ce caracterizează gradul de nedeterminare asupra unui sistem. În teoria informație ***entropia*** unei ***surse discrete fără memorie*** (***SDFM***) este definită ca fiind cantitatea medie de informație furnizată de apariția unui simbol a generat de sursă. Entropia unei *SDMF* se notează cu $H(S)$, unde S este o *SDMF* și are următoarea expresie:

$$H(S) = \sum_{i=1}^{m} p(A_i) \cdot \log_2 \frac{1}{p(A_i)} = -\sum_{i=1}^{m} p(A_i) \cdot \log_2 p(A_i)$$

unde m reprezintă numărul de simboluri ale sursei S, A_i ($1 \le i \le m$) reprezintă simbolurile sursei S; $p(A_i)$ reprezintă frecvența (probabilitatea) de apariție a simbolului A_i. Ca și cantitatea de informație, entropia se măsoară în biți.

Entropia maximă a unei surse este $H_{max}(S) = \log_2 m$, unde m reprezintă numărul de simboluri ale sursei S.

Se poate observa foarte ușor că entropia unei surse verifică următoarele relații:

$$0 \le H(S) \le H_{max}(S),$$

$$H(S) = H_{max}(S) \Leftrightarrow p(A_i) = 1/m, \forall\ 1 \le i \le m.$$

Eficiența unei surse se definește ca fiind raportul dintre entropia sursei și entropia maximă a sursei:

$$\eta_S = H(S) / H_{max}(S).$$

Redundanța unei surse reprezintă excesul de informație față de strictul necesar și este dată de diferența dintre entropia maximă a sursei și entropia "reală" a sursei:

$$R_S = H_{max}(S) - H(S).$$

Redundanța relativă a unei surse se definește ca fiind raportul dintre redundanța sursei și entropia maximă a sursei:

$$\rho_S = R(S) / H_{max}(S).$$

Algoritmi de compresie a datelor

Un compresor de date este o aplicație care, pe baza unuia sau mai multor algoritmi de compresie, diminuează spațiul necesar stocării informației utile conținute de un anumit set de date. Pentru orice compresor de date este necesară condiția de existență a cel puțin unui decompresor care, pe baza informațiilor furnizate de compresor, să poată reconstitui informația care a fost comprimată. În cazul în care nu există un decompresor, atunci datele comprimate devin inutile pentru utilizator deoarece acesta nu mai are acces la informația stocată în arhivă (o ***arhivă*** reprezintă rezultatul obținut în urma utilizării unui compresor).

Un compresor de date este format din următoarele elemente:

- una sau mai multe surse de informație;
- unul sau mai mulți algoritmi de compresie.
- una sau mai multe transformări.

Sursa de informație pentru un compresor poate fi constituită de unul sau mai multe fișiere sau de un flux de date care a fost transmis compresorului prin intemediul unei rețele. Datele arhivate urmează să fie stocate într-o arhivă care urmează să fie păstrată local sau urmează să fie transmisă mai departe.

Un model simplu de algoritm care poate sta la baza unui compresor de date este următorul:

1. *Fie S sursa de informație*
2. *Fie D destinația datelor comprimate*
3. *Fie A algoritmul de compresie utilizat*
4. *Se inițializează algoritmul de compresie*
5. *Pentru fiecare simbol i furnizat de sursa S execută:*
   5.1 *Fie R rezultatul obținut prin codificara simbolului i folosind algoritmul A*
   5.2 *Dacă rezultatul R nu este vid, transmite R către D*
6. *Transmite algoritmului A faptul că s-a încheiat recep-*

ționarea de simboluri de la sursa S și fie R ultimul rezultat reținut de algoritmul de compresie

7. Dacă rezultatul R nu este vid, transmite R către D

Semnificația iterației 5.2 este următoarea: în cazul în care algorimtul de compresie nu codifică simbolurile individual ci codifică un șir de simboluri folosind un singur identificator, atunci rezultatul obținut la iterația 5.1 este vid atât timp cât nu s-a ajuns la numărul de simboluri necesare pentru furnizarea unui rezultat. Pentru a optimiza această clasă de algorimi de compresie, propozițiile de la iterația 5, respectiv iterația 5.1 pot fi înlocuite cu "*pentru fiecare set de n simboluri furnizate de sursa S*", respectiv "*fie R rezultatul obținut prin codificara setului de n simboluri folosind algoritmul A*".

La un moment dat, algoritmul de compresie așteaptă apariția unui număr de simboluri pentru a putea furniza un rezultat, dar sursa de informație nu mai emite simboluri. Datorită acestui fapt, pentru a nu se pierde informație, este necesar ca algoritmului de compresie să i se comunice terminarea compresiei și acesta la rândul său trebuie să furnizeze ultimul rezultat obținut pe baza simbolurilor pe care nu le-a codificat. În cazul formatelor de date multimedia, o transformare poate fi identificată cu un cuantificator și poate duce la pierderea anumitor date (de exemplu, un punct gri pe o imagine complet albă poate deveni alb deoarece nu este perceput de ochiul uman).

De obicei, compresoarele de date avansate (cum sunt aplicațiile *RAR*, *ACE*, *BZIP* etc.) folosesc o înlănțuire de algoritmi similari celui prezentat anterior cu precizarea că în afară de primul algoritm din înlănțuire, sursa de informații pentru un alt algoritm este constituită de rezultatul furnizat de algoritmul anterior.

Una dintre cele mai importante întrebări care se pun cu privire la compresie datelor este: "*Cum să facem ca datele să se comprime mai bine?*"

Pentru anumite seturi de date particulare s-au găsit răspunsuri, dar despre aceasta vom vorbi în detaliu în alt episod. Pentru a face datele "*mai comprimabile*" se aplică transformări. Pentru aplicarea unei transformări, în algoritmul prezentat anterior se înlocuiește algoritmul de compresie *A* cu transformarea *T* și se folosește în combinație cu algoritmii de compresie.

În cele ce urmează vom prezenta câțiva algoritmi folosiți pentru comprimarea datelor.

Există două categorii principale de algoritmi pentru compresia datelor:

- algoritmi de compresie fără pierderi de date
  - algoritmi entropici de compresie a datelor (algoritmi care elimină redundanța din cadrul seturilor de date)
    - algoritmul *Huffman*
    - algoritmi de *compresie aritmetică* (acesta este cel mai performant algoritm entropic cunoscut).
  - algoritmi bazați pe dicționare
    - *Lempel Ziff* 77 (*LZ*77), *Lempel Ziff* 78 (*LZ*78)
    - variante ale algoritmilor din categoria *LZ*
  - transformări fără pierderi de date
    - *Run Length Encoding* (*RLE*)
    - *Burrow-Wheeler Transform* (*BWT*)
    - *transformarea delta*
- algoritmi de compresie cu pierderi de date
  - algoritmi folosiți pentru compresia audio care se bazează pe proprietățile undelor sonore și perceperea lor de către om;
  - algoritmi utilizați pentru compresia imaginilor digitale
  - algoritmi utilizați pentru compresia datelor video.

Algoritmii de compresie aparținând celei de-a doua clase (cea cu pierderi de date) se folosesc împreună cu cei din prima pentru atingerea unor rate mari de compresie.

Algoritmul Huffman

Ideea care stă la baza acestui algoritm este aceea de a atașa coduri de lungime mică (exprimată în biți) simbolurilor furnizate de o sursă de informație și a căror frecvență de apariție este mare, și de a atașa coduri de lungime mai mare simbolurilor a căror frecvență de apariție este mică.

În concluzie, folosind acest algoritm se reduce lungimea medie a codurilor folosite pentru a reprezenta simbolurile alfabetului.

Codurile generate de acest algoritm sunt optime în cazul în care toate frecvențele de apariție ale simbolurilor sunt puteri întregi ale lui 1/2.

Algoritmul lui *Huffman* constă în construirea unui arbore binar în care fiecare nod are fie nici un fiu, fie exact 2 fii, în care frunzele reprezintă simbolurile generate de sursa de informație și au atașat un cost egal cu frecvența de apariție a acestora, iar nodurile interne au atașat un cost egal cu suma costurilor fiilor. Prin urmare, rădăcina arborelui are costul 1.

Atunci când se cunosc frecvențele de apariție ale simbolurilor generate de o sursă de informație *S* și faptul că sursa a generat în total *m* simboluri distincte ale unui alfabet, construcția arborelui binar devine foarte simplă.

Primul pas constă în construirea unei păduri formată din *m* arbori. Fiecare arbore din pădure are un singur element care conține un simbol și frecvența lui de apariție. Toți cei *m* arbori din pădure sunt distincți (nu există doi arbori al cărui element să conțină același simbol).

În continuare, la fiecare pas se elimină câte doi arbori din pădure pentru care costul rădăcinii este minim. Acești doi arbori vor fi subarborii unui nod al cărui cost va fi egal cu suma frecvențelor celor doi arbori eliminați din pădure. Arborele nou creat se adaugă în pădure.

Algoritmul se termină în momentul în care în pădure mai rămâne un singur arbore, adică după efectuarea a *m* - 1 iterații.

Din analiza acestui algoritm rezultă faptul că drumul de la rădăcină până la frunzele care au costul mai mic este

mai mare decât drumul de la rădăcină până la frunzele care au costul mai mare.

În figura 1 se poate observa modul de construire al arborelui pentru șirul "*To be or not to be*" generat de o sursă de informație *S*. Șirul are lungimea $n = 18$ și $m = 8$ simboluri distincte. Simbolurile distincte sunt "*b*", "*e*", "*n*", "*o*", "*r*","*t*","*T*" respectiv "_"(spațiu) și au frecvențele de apariție 2/18, 2/18, 1/18, 4/18, 1/18, 2/18, 1/18, 5/18.

Datorită faptului că toate fracțiile au același numitor, îi vom elimina, deci la sfârșitul algoritmului rădăcina arborelui obținut va avea costul *n*, unde *n* este numărul total de simboluri generate de sursa *S*. Eliminând numitorul comun se optimizează execuția unui program care implementează acest algoritm datorită faptului că operațiile cu numere întregi sunt executate mai rapid decât operațiile cu numere reale.

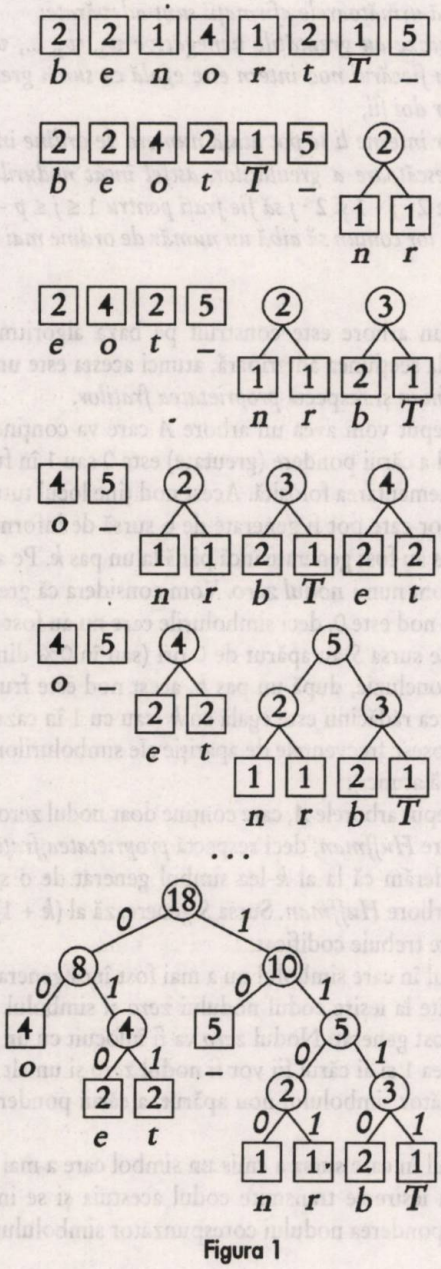


Figura 1

Lungimea codului pentru un simbol generat de sursa *S* va fi egală cu lungimea drumului de la frunza corespunzătoare unui simbol până la rădăcina arborelui obținut folosind algoritmul lui *Huffman*.

În arborele obținut în final, vom eticheta fiecare muchie dintre două noduri cu 0 dacă nodul fiu se află în stânga părintelui și cu 1 dacă nodul fiu se află în dreapta. Pentru a codifica simbolurile generate de sursa *S*, se va parcurge șirul generat și pentru fiecare simbol se va transmite la o ieșire *E* drumul de la rădăcină până la frunza corespunzătoare simbolului curent astfel: se transmit la ieșire etichetele muchiilor întâlnite pe drum.

Simbolurile din șirul dat anterior ca exemplu au codurile:

- "b" 1110
- "e" 010
- "n" 1100
- "o" 00
- "r" 1101
- "t" 011
- "T" 1111
- "_" 10

Având codurile simbolurilor, șirul din exemplu se codifică astfel:
11110010111001010001101101100000111001100101110010

Lungimea codificării exemplului este de 50 biți (7 *bytes*). Dacă sursa de informație folosește alfabetul *ASCII*, atunci lungimea în bytes a mesajului este 18, deci folosind această metodă se diminuează spațiul utilizat cu 11 bytes. Pentru a putea decodifica acest mesaj, trebuie transmis și arborele creat, sau informațiile necesare pentru a putea fi construit. Dacă aceste informații lipsesc, nu se poate reconstitui mesajul inițial.

Arborele de codificare obținut cu agoritmul lui *Huffman* poartă numele de *dicționar*. *Dicționarul* este un termen general după cum vom vedea în cazul altor tehnici de compresie. ***Dicționarul*** desemnează orice structură de date (sau informațiile necesare construirii ei) care se atașează șirului comprimat pentru ca programul de decompresie să poată reconstitui informația inițială.

În cazul acestui algoritm o reprezentare eficientă a arborelui are dimensiunea de $2 \cdot m$ biți.

Există mai multe metode de a transmite dicționarul. Cel mai bine este ca acesta să fie și el comprimat pentru a elimina o parte din redundanță.

Pentru a determina care este redundanța rămasă după compresie, trebuie mai întâi calculată entropia șirului de intrare care este $H = 2{,}747167$ biți, deci ar fi trebuit să se folosească în medie *H* biți pentru a codifica fiecare simbol. Acest lucru nu este posibili deoarece algoritmul lui *Huffman* atașează un număr întreg de biți pentru fiecare caracter în parte. Lungimea medie a codului unui simbol este 2,777778 biți deci pentru fiecare simbol se transmite o redundanță de aproximativ 0,23 biți la care se adaugă redundanța dată de transmiterea dicționarului.

Entropia maximă a oricărui șir obținut din simbolurile alfabetului *ASCII* este $H_{max} = \log_2 m = \log_2 256 = 8$ biți, unde $m = 256$ este mărimea alfabetului, deci orice simbol *ASCII* se poate reprezenta folosind maxim 8 biți.

În acest caz, lungimea medie a unui cod este mai mică decât entropia maximă, deci șirul se poate comprima.

În cazul în care frecvențele de apariție ale simbolurilor sunt egale, iar numărul de simboluri distincte generate de o sursă *S* este egal cu numărul de simboluri ale alfabetului, atunci șirul simbolurilor nu se poate comprima folosind această metodă în felul cum a fost prezentată mai sus, deoarece lungimea medie a codului unui simbol devine egală cu entropia maximă la care se mai adaugă și informația conținută de dicționar; în concluzie, cantitatea de informație transmisă va fi mai mare decât cea inițială. În acest caz, se folosesc variante mai complexe ale algoritmului.

Variante ale algoritmului lui Huffman

Există trei variante des aplicate ale acestui algoritm:

- algoritmul *Huffman static*;
- algoritmul *Huffman semi-static*;
- algoritmul *Huffman dinamic*.

Diferențele dintre cele trei variante sunt următoarele:

- în cazul variantei statice, atât compresorul, cât și decompresorul dețin același arbore de compresie calculat pe baza unor frecvențe fixe de apariție și nu mai este necesară calcularea unui arbore nou și nici transmiterea acestuia. Dezavantajul acestei metode este că, dacă frecvențele de apariție ale simbolurilor generate de o sursă diferă foarte mult de cele fixe utilizate, atunci s-ar putea ca pentru simboluri cu frecvență mare de apariție să fie transmise coduri foarte lungi și astfel cantitatea de informație comprimată poate să depășească cu mult cantitatea de informație care a fost generată de o sursă.
- varianta semi-statică utilizează algoritmul de construire a arborelui de compresie prezentat anterior. Are ca dezavantaj faptul că șirul simbolurilor generate de o sursă de informație trebuie parcurse de două ori, o dată pentru calcularea frecvențelor necesare construirii arborelui și o dată pentru codificarea șirului simbolurilor; în cazul în care simbolurile au aproximativ aceeași frecvență de apariție, algoritmul nu oferă o compresie bună. Parcurgerea de două ori a șirului simbolurilor generate de o sursă de informație este un incovenient deoarece acesta trebuie stocat, iar dimensiunea datelor care trebuie comprimate, în zilele noastre, este foarte mare și calculatoarele personale (de cele mai multe ori) nu dețin resursele necesare stocării datelor.
- pentru varianta *dinamică* există doi algoritmi performanți: *FGK (Faller, Gallager, Knuth)* și *V (Vitter)*. Aceștia au în comun faptul că arborele se construiește dinamic pe măsură ce o sursă de informație generează simboluri, deci este necesară o singură parcurgere a șirului simbolurilor și nu este necesară stocarea lor. Doar o mică parte dintre ele sunt stocate cu scopul de a optimiza procesul de compresie. Rata de compresie a acestor doi algoritmi variază. În anumite cazuri rezultatele obținute sunt cu mult mai bune decât cele date de varianta statică, dar în cazul cel mai rău pentru varianta statică, rezultatele variantelor dinamice sunt optime de cele mai multe ori.

Primele două variante nu mai necesită explicații suplimentare, așadar, în continuare, vom prezenta în detaliu varianta dinamică ***FGK***, varianta dinamică ***Vitter*** fiind similară.

Algoritmul FGK

Algoritmul *FGK* precum și algoritmul *V* se bazează pe *proprietatea fraților* enunțată și demonstrată de *Gallager* în anul 1978:

"Un arbore binar cu p frunze care au greutăți (în cazul de față frecvențe) nenegative este arbore Huffman dacă și numai dacă următoarele afirmații sunt adevărate:

- *cele p frunze au greutățile nenegative w_1, w_2, ..., w_p și greutatea fiecărui nod intern este egală cu suma greutăților celor doi fii;*
- *nodurilor interne li se pot atașa numere de ordine în ordinea crescătoare a greutăților, astfel încât nodurile cu numerele $2 \cdot j - 1$ și $2 \cdot j$ să fie frați pentru $1 \le j \le p - 1$ și părintele lor comun să aibă un număr de ordine mai mare."*

Dacă un arbore este construit pe baza algoritmului prezentat la secțiunea anterioară, atunci acesta este un arbore *Huffman* și respectă *proprietatea fraților*.

La început vom avea un arbore *A* care va conține un singur nod a cărui pondere (greutate) este 0 sau 1 în funcție de implementarea folosită. Acest nod ține locul tuturor simbolurilor care pot fi generate de o sursă de informație *S* și care nu au fost generate încă până la un pas *k*. Pe acest nod și îl vom numi ***nodul zero***. Vom considera că greutatea acestui nod este 0, deci simbolurile care nu au fost încă generate de sursa *S* au apărut de 0 ori (sau în 0% din cazuri). În concluzie, după un pas *k*, acest nod este frunză, iar ponderea rădăcinii este egală cu *k* (sau cu 1 în cazul în care se folosesc frecvențele de apariție ale simbolurilor generate până atunci).

La început arborele *A*, care conține doar nodul zero, este un arbore *Huffman*, deci respectă *proprietatea fraților*.

Considerăm că la al *k*-lea simbol generat de o sursă avem un arbore *Huffman*. Sursa *S* generează al $(k + 1)$-lea simbol care trebuie codificat.

În cazul în care simbolul nu a mai fost încă generat, se va transmite la ieșire codul nodului *zero* și simbolul care tocmai a fost generat. Nodul *zero* va fi înlocuit cu un nod cu ponderea 1 și ai cărui fii vor fi nodul *zero* și un alt nod corespunzător simbolului nou apărut, a cărui pondere va fi tot 1.

În cazul în care sursa a emis un simbol care a mai fost generat, la ieșire se transmite codul acestuia și se incrementează ponderea nodului corespunzător simbolului generat.

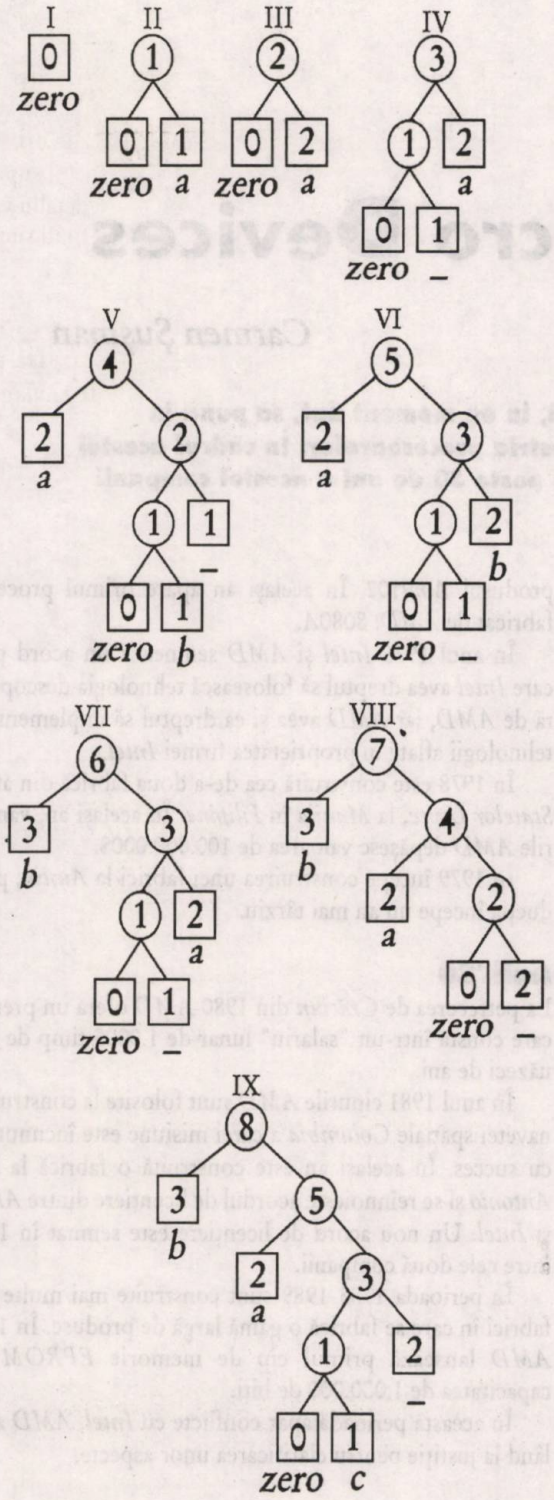


Figura 2

Codurile simbolurilor din arbore sunt construite la fel ca în cazul variantelor *semi-statice*, adică sunt date de drumul parcurs de la rădăcină până la frunza corespunzătoare simbolului care trebuie codificat.

În continuare trebuie actualizate ponderile celorlalte noduri interne deaorece după adăugarea unui nod nu mai este respectată *proprietatea fraților* fiindcă ponderea rădăcinii nu mai este egală cu suma ponderilor frunzelor subarborilor ei.

Aceste două cazuri de apariție ale unui simbol sunt tratate în mod identic de algoritmii *FGK* și *V*.

În continuare vom prezenta modul în care se actualizează ponderile nodurilor interne în cadrul algoritmului *FGK*.

Pentru a menține proprietatea fraților, trebuie parcurs drumul de la frunza actualizată ultima dată până la rădăcină. Ponderea fiecărui nod din drum trebuie incrementată cu 1 pentru a fi respectată prima condiție a proprietății fraților, iar pentru a doua condiție, la fiecare pas trebuie interschimbat nodul curent cu un nod din arbore care are aceeași pondere cu cea a nodului curent și care are cel mai mare număr de ordine. Părintele nodului curent devine părintele noului nod și invers. Trebuie avut în vedere faptul că un nod nu se poate interschimba cu un strămoș de-al său.

În figura alăturată este prezentat modul de construcție și actualizare a arborelui folosind algoritmul *FGK*, pentru șirul "*aa bbb c*", unde simbolul *spațiu* este reprezentat prin "_".

La fiecare pas, arborii din figură sunt arbori *Huffman*, deci respectă și a doua condiție a proprietății fraților. Pentru a verifica acest lucru se numerotează nodurile din arbore în ordine creascătoare începând de la ultimul nivel până la primul și de la stânga la dreapta.

Algoritmul V

Algoritmul *V* diferă de algoritmul *FGK* prin faptul că, în momentul în care se realizează actualizarea arborelui, se încearcă minimizarea expresiilor **SUM**(l_i) și **MAX**(l_i), unde **SUM**(l_i) reprezintă suma lungimilor drumurilor de la rădăcină până la frunze, iar **MAX**(l_i) reprezintă lungimea drumului de la rădăcină până la cea mai îndepărtată frunză. Cu alte cuvinte, se încearcă minimizarea adâncimii maxime a arborelui și a lungimii drumului extern.

Complexitatea celor doi algoritmi este aceeași, dar algoritmul *V* este mai performant în cazul în care probabilitățile de apariție ale simbolurilor sunt aproximativ aceleași.

Jeffrey S. Vitter a demonstrat faptul că în cel mai rău caz, algoritmul *V* transmite la fiecare simbol un bit în plus față de metoda *semi-statică*, în timp ce algoritmul *FGK* transmite în cel mai rău caz de două ori mai mulți biți pe simbol relativ la metoda *semi-statică*.

În episodul următor...

Algoritmii de compresie *Huffman* atașează câte un cod fiecărui simbol generat de o sursă *S*. În numărul următor vom prezenta o metodă de compresie care codifică un șir de simboluri cu ajutorul unui singur cod. Este vorba despre cel mai performat algoritm entropic de compresie și anume algoritmul de *compresie aritmetică*.

Claudiu Soroiu este redactor al GInfo. Poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

Advanced Micro Devices

Carmen Şuşman

Compania AMD a fost singura care a reuşit, la un moment dat, să pună în pericol supremaţia gigantului Intel în industria procesoarelor. În cadrul acestui articol vă vom prezenta pe scurt istoria de peste 30 de ani a acestei companii americane.

La început

În anul 1968 *Jerry Sanders* a renunţat la funcţia sa de director comercial în cadrul companiei *Fairchild Semiconductor* şi s-a văzut pus în situaţia de a conduce o echipă de entuziaşti al căror scop era crearea unei companii care să aibă un cuvânt de spus în industria semiconductoarelor.

Pe data de 1 mai 1969 cei opt membri ai echipei au reuşit să creeze în mod oficial compania *AMD* al cărei sediu era în sufrageria lui *John Carey*, unul dintre fondatori; capitalul companiei era de 100.000$. La scurt timp sediul s-a mutat în spatele unei fabrici de covoare din *Santa Clara*. În septembrie, *AMD* a reuşit să strângă suficiente fonduri pentru a începe producţia şi s-a mutat în primul său sediu permanent (în *Sunnyvale*).

În primii ani, majoritatea produselor *AMD* erau simple copii îmbunătăţite ale produselor oferite de alte companii. Primul produs ieşit din fabrica *AMD* (în noiembrie 1969) a fost *Am*9300, un registru de deplasare pe patru biţi. La sfârşitul primului an de activitate compania avea 53 de angajaţi, oferea 18 tipuri de produse, dar nu efectuase încă nici o vânzare.

Primul produs asupra căruia *AMD* avea toate drepturile de proprietate a fost *Am*2501, care a apărut în 1970.

În septembrie 1972 *AMD* devine cotată la bursă fiind lansate 525.000 de acţiuni cu o valoare nominală de 15$ fiecare.

În continuare compania se dezvoltă foarte repede, în ianuarie 1973 începând producţia la prima fabrică *AMD* din afara *Statelor Unite* (la *Penang*, în *Malayesia*).

În 1974, după cinci ani de activitate, *AMD* avea vânzări în valoare de 26.500.000$, avea peste 1500 de angajaţi şi oferea peste 200 de tipuri de produse.

Următorii cinci ani

Pentru a aniversa primii cinci ani de activitate, în mai 1974 *AMD* a organizat o petrecere publică în cadrul căreia au fost oferite ca premii un televizor şi zece biciclete.

În 1975 *AMD* intră pe piaţa memoriilor *RAM*, lansând produsul *Am*9102. În acelaşi an apare primul procesor fabricat de *AMD*: 8080*A*.

În anul 1976 *Intel* şi *AMD* semnează un acord prin care *Intel* avea dreptul să folosească tehnologia descoperită de *AMD*, iar *AMD* avea şi ea dreptul să implementeze tehnologii aflate în proprietatea firmei *Intel*.

În 1978 este construită cea de-a doua fabrică din afara *Statelor Unite*, la *Manila* în *Filipine*. În acelaşi an, vânzările *AMD* depăşesc valoarea de 100.000.000$.

În 1979 începe construirea unei fabrici la *Austin*; producţia începe un an mai târziu.

Anii '80

La petrecerea de *Crăciun* din 1980 *AMD* oferă un premiu care constă într-un "salariu" lunar de 1.000$ timp de douăzeci de ani.

În anul 1981 cipurile *AMD* sunt folosite la construirea navetei spaţiale *Columbia* a cărei misiune este încununată cu succes. În acelaşi an este construită o fabrică la *San Antonio* şi se reînnoieşte acordul de licenţiere dintre *AMD* şi *Intel*. Un nou acord de licenţiere este semnat în 1982 între cele două companii.

În perioada 1983-1989 sunt construite mai multe noi fabrici în care se fabrică o gamă largă de produse. În 1986 *AMD* lansează primul cip de memorie *EPROM* cu capacitatea de 1.000.000 de biţi.

În această perioadă apar conflicte cu *Intel*, *AMD* apelând la justiţie pentru clarificarea unor aspecte.

Anii '90

În martie 1991 *AMD* introduce familia de procesoare *Am*386, întrerupând monopolul companiei *Intel* pe această piaţă. Până în octombrie sunt vândute peste 1.000.000 de astfel de procesoare.

În februarie 1992 justiţia se pronunţă în favoarea *AMD* în conflictul cu *Intel*; *AMD* a primit toate drepturile de fabricare şi vânzare asupra procesoarelor din familia *Am*386.

În aprilie 1993 apar primele procesoare din familia

*Am*486; în același an sunt anunțate planurile pentru familia de procesoare *K*5 care apare după câțiva ani.

În ianuarie 1994 se semnează un acord pe termen lung cu *Compaq* prin care procesoarele *Am*486 sunt folosite la fabricarea calculatoarelor *Compaq*.

Pe data de 10 martie 1994 se încheie un nou proces cu *Intel*; prin decizia din această dată *AMD* are dreptul să utilizeze microoperațiile folosite în coprocesorul matematic *Intel* 287.

Pe data de 1 mai 1994 se sărbătorește aniversarea de 25 de ani a companiei.

Familia de procesoare *K*6 apare în anul 1997 și pune serios în pericol dominația companiei *Intel* pe piața procesoarelor. Totuși, *AMD* devine lider doar în 1999 când este lansat procesorul *AMD Athlon*.

În această perioadă *AMD* achiziționează sau fuzionează cu o serie de companii. Scopul declarat era de a revitaliza industria procesoarelor prin oferirea de produse mai performante decât cele fabricate de *Intel*.

Ultima perioadă

Sfârșitul anului 1999 și începutul anului 2000 au marcat întreruperea dominației companiei *Intel* pe piața procesoarelor. Noul procesor *AMD Athlon* depășea performanțele procesoarelor *Pentium III* oferite de *Intel*. Totul a culminat pe data de 6 martie 2000 când *AMD* a anunțat disponibilitatea imediată a procesoarelor *Athlon* cu frecvența de 1 GHz. Deși doar după două zile *Intel* a anunțat că oferă procesoare *Pentium III* cu frecvența de 1 GHz, acestea nu au fost disponibile decât în august. Acest eveniment a avut ca urmare o creștere spectaculoasă a vânzărilor companiei *AMD* care a avut venituri de peste un miliard de dolari în primul trimestru al anului.

Poziția companiei *AMD* s-a întărit în vara anului 2000 când *Intel* a lansat procesorul *Pentium III* cu frecvența de 1,13 GHz. La scurt timp după această lansare procesoarele au fost retrase de pe piață datorită unor defecte semnalate de presă. Aceasta a dus la un nou val de cereri pentru procesoarele *AMD* care au atins frecvențe din ce în ce mai mari.

Primul semn al revenirii celor de la *Intel* a fost lansarea procesoarelor *Pentium* 4 care aveau frecvențe mult mai mari, deși performanțele erau mai scăzute. Până la urmă frecvențele și-au spus cuvântul, iar *Intel* a redevenit lider mondial în momentul în care a reușit să depășească frecvența de 2 GHz. Între timp, *AMD* și-a îmbunătățit seria de produse lansând în 2001 procesorul *Athlon XP* care era mai performant decât un procesor *Intel* cu frecvența mai mare cu câteva sute de megaherți.

Chiar dacă acum *AMD* este numărul 2 în industria procesoarelor, anul 2003 s-ar putea să ne aducă o serie de noutăți. *AMD* a anunțat faptul că va lansa primul procesor pe 64 de biți care este compatibil și cu aplicațiile actuale pe 32 de biți. Varianta pentru *server*-e va purta denumirea *AMD Opteron*, iar cea pentru calculatoarele de birou va fi numită *AMD Athlon* 64. Cu siguranță *Intel* nu se va lăsa mai prejos și va lansa o serie de produse noi. Din acest conflict vor câștiga doar cumpărătorii, deoarece cele două companii vor fi nevoite să scadă prețurile pentru a acapara un segment cât mai mare de piață.

Încheiere

Pornind de la cei 100.000 de dolari pe care îi avea la dispoziție în 1969, compania *AMD* s-a dezvoltat și a ajuns să aibă acum venituri anuale de 4,6 miliarde de dolari.

Cu toate acestea, au existat și perioade mai grele datorate fie recesiunii economice, fie competiției cu care se confrunta. Deseori, *AMD* a fost nevoită să concedieze o parte din angajați pentru a supraviețui.

Chiar și în acest moment este evident faptul că *AMD* se află în dificultate. Compania înregistrează pierderi în ultima perioadă, iar recent au fost concediați 15% dintre angajați.

Cu toate acestea existența companiei *AMD* este de bun augur pentru piața procesoarelor. Întotdeauna *AMD* a oferit produse mai ieftine decât *Intel* la performanțe apropiate. În perioada 1999-2000 procesoarele *AMD* erau chiar mai performante decât cele oferite de *Intel*, chiar dacă prețurile erau mult mai mici.

Se observă destul de clar faptul că utilizatorii români preferă să achiziționeze procesoare *Athlon XP* cu frecvențe mai mici de 2 GHz, chiar dacă *Intel* oferă produse mai performante. Evident, principalul criteriu care duce la luarea unei astfel de decizii este prețul mult mai mic al procesoarelor *AMD*.

Se știe că, la ora actuală, un procesor *AMD* este mai performant decât un procesor *Intel* care funcționează la aceeași frecvență. Din nefericire, frecvențele procesoarelor *AMD* nu ating în acest moment decât 2,25 GHz, în timp ce *Intel* a depășit recent bariera de 3 GHz. Pentru a arăta faptul că "frecvența nu este totul" *AMD* a introdus un indicator de performanță pentru procesoarele sale. Astfel, teoretic, un *AMD Athlon XP* 2800+ (cel mai performant procesor *AMD*) ar trebui să ofere performanțe asemănătoare cu cele oferite de un *Intel Pentium* 4 cu frecvența de 2,8 GHz, chiar dacă frecvența reală a procesorului *AMD* este cu peste 500 MHz mai mică.

Introducerea acestor indicatori a fost îndelung discutată datorită confuziilor pe care le creează. Totuși, se pare că *AMD* va păstra acești indicatori și pentru familia următoare de procesoare.

În acest moment *AMD* deține un segment de piață destul de mare pentru procesoarele destinate calculatoarelor de birou, unul mult mai mic pentru procesoarele destinate calculatoarelor portabile și unul aproape nesemnificativ pentru procesoarele destinate *server*-elor. Există șanse (relativ mici) ca situația să se schimbe în 2003; nu putem decât să așteptăm!

Carmen Șușman este redactor-șef adjunct al GInfo și poate fi contactată prin e-mail la adresa **schummy@email.ro**.

Evoluția unui PROGRAMATOR

Vă prezentăm în cadrul acestui articol modul în care vor arăta programele voastre simple pe măsura trecerii timpului.

Elev de gimnaziu

```
10 PRINT "HELLO WORLD"
20 END
```

Elev de liceu

```
Program Hello;
Begin
  Writeln('Hello World')
End.
```

Student

```
(defun hello
  (print
    (cons 'Hello (list 'World))))
```

Programator începător

```
#include <stdio.h>
void main(void){
  char *message[] = {"Hello ", "World"};
  int i;
  for(i = 0; i < 2; ++i)
    printf("%s", message[i]);
  printf("\n");
}
```

Programator ocazional

```
#include <iostream.h>
#include <string.h>

class string{
  private:
    int size;
    char *ptr;
  public:
    string(): size(0), ptr(new char('\0'))
    {}
    string(const string &s) : size(s.size){
      ptr = new char[size + 1];
      strcpy(ptr, s.ptr);
    }
    ~string(){
      delete[] ptr;
    }
    friend ostream &operator <<(ostream &,
                                const string &);
    string &operator = (const char *);
};

ostream &operator <<(ostream &stream,
                     const string &s){
  return (stream << s.ptr);
}

string &string::operator=(const char *chrs){
  if (this != &chrs){
    delete[] ptr;
    size = strlen(chrs);
    ptr = new char[size + 1];
    strcpy(ptr, chrs);
  }
  return (*this);
}

int main(){
  string str;

  str = "Hello World";
  cout << str << endl;
  return(0);
}
```

```
[uuid(2573F8F4-CFEE-101A-9A9F-
                        00AA00342820)]
library LHello{
   importlib("actimp.tlb");
   importlib("actexp.tlb");

   #include "pshlo.idl"

   [uuid(2573F8F5-CFEE-101A-9A9F-
                        00AA00342820)]
   cotype THello{
      interface IHello;
      interface IPersistFile;
   };
};

[exe, uuid(2573F890-CFEE-101A-9A9F-
                        00AA00342820)]
module CHelloLib{
   importheader(<windows.h>);
   importheader(<ole2.h>);
   importheader(<except.hxx>);
   importheader("pshlo.h");
   importheader("shlo.hxx");
   importheader("mycls.hxx");

   importlib("actimp.tlb");
   importlib("actexp.tlb");
   importlib("thlo.tlb");

   [uuid(2573F891-CFEE-101A-9A9F-
            00AA00342820),aggregatable]
   coclass CHello{
      cotype THello;
   };
};

#include "ipfix.hxx"
extern HANDLE hEvent;
class CHello: public CHelloBase{
   public:
      IPFIX(CLSID_CHello);
      CHello(IUnknown *pUnk);
      ~CHello();
      HRESULT __stdcall PrintSz(LPWSTR
                              pwszString);

   private:
      static int cObjRef;
};

#include <windows.h>
#include <ole2.h>
#include <stdio.h>
#include <stdlib.h>
#include "thlo.h"
#include "pshlo.h"
#include "shlo.hxx"
#include "mycls.hxx"

int CHello::cObjRef = 0;

CHello::CHello(IUnknown *pUnk) :
                        CHelloBase(pUnk){
   cObjRef++;
   return;
}

HRESULT __stdcall CHello::PrintSz(LPWSTR
                              pwszString){
   printf("%ws\n", pwszString);
   return(ResultFromScode(S_OK));
}
CHello::~CHello(void){
   cObjRef--;
   if( cObjRef == 0)
      PulseEvent(hEvent);
   return;
}

#include <windows.h>
#include <ole2.h>
#include "pshlo.h"
#include "shlo.hxx"
#include "mycls.hxx"

HANDLE hEvent;

int _cdecl main(int argc, char *argv[]){
   ULONG ulRef;
   DWORD dwRegistration;
   CHelloCF *pCF = new CHelloCF();

   hEvent = CreateEvent(NULL, FALSE, FALSE,
                              NULL);
   CoInitializeEx(NULL,
                  COINIT_MULTITHREADED);
   CoRegisterClassObject(CLSID_CHello,
   pCF, CLSCTX_LOCAL_SERVER, REGCLS_MULTI-
                              PLEUSE,
                        &dwRegistration);
   WaitForSingleObject(hEvent, INFINITE);
   CoRevokeClassObject(dwRegistration);
   ulRef = pCF->Release();
   CoUninitialize();
   return(0);
}

extern CLSID CLSID_CHello;
extern UUID LIBID_CHelloLib;
CLSID CLSID_CHello = {
   0x2573F891,
   0xCFEE,
   0x101A,
   {0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34, 0x28,
                              0x20}
};
UUID LIBID_CHelloLib = {
   0x2573F890,
   0xCFEE,
   0x101A,
   {0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34, 0x28,
                              0x20}
};
```

```
#include <windows.h>
#include <ole2.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "pshlo.h"
#include "shlo.hxx"
#include "clsid.h"

int _cdecl main(int argc,char *argv[]){
  HRESULT hRslt;
  IHello *pHello;
  ULONG ulCnt;
  IMoniker *pmk;
  WCHAR wcsT[_MAX_PATH];
  WCHAR wcsPath[2 * _MAX_PATH];

  wcsPath[0] = '\0';
  wcsT[0] = '\0';
  if(argc > 1){
    mbstowcs(wcsPath, argv[1],
                    strlen(argv[1]) + 1);
    wcsupr(wcsPath);
  }
  else{
    fprintf(stderr, "Object path must be
                          specified\n");
              return(1);
}

  if(argc > 2)
    mbstowcs(wcsT, argv[2],
                    strlen(argv[2]) + 1);
  else
    wcscpy(wcsT, "Hello World");

  printf("Linking to object %ws\n",
                              wcsPath);
  printf("Text String %ws\n", wcsT);
  hRslt = CoInitializeEx(NULL,
                    COINIT_MULTITHREADED);
 if(SUCCEEDED(hRslt)){
    hRslt = CreateFileMoniker(wcsPath,
                                &pmk);
    if(SUCCEEDED(hRslt))
      hRslt = BindMoniker(pmk, 0,
          IID_IHello, (void **)&pHello);

    if(SUCCEEDED(hRslt)){
      pHello->PrintSz(wcsT);
      Sleep(2000);
      ulCnt = pHello->Release();
    }
    else
      printf("Failure to connect, status:
                          %lx", hRslt);
    CoUninitialize();
  }
  return (0);
}
```

Hacker începător

```
#!/usr/local/bin/perl
$msg="Hello, world.\n";
if ($#ARGV >= 0){
  while(defined($arg=shift(@ARGV))){
    $outfilename = $arg;
    open(FILE, ">" . $outfilename) ||
          die "Can't write $arg: $!\n";
    print (FILE $msg);
    close(FILE) || die "Can't close $arg:
                                $!\n";
  }
}
else{
  print ($msg);
}
```

Hacker expert

```
#include <stdio.h>
#define S "Hello, World\n"
main(){exit(printf(S) == strlen(S) ? 0 : 1);}
```

Hacker ocazional

```
% cc -o a.out ~/src/misc/hw/hw.c
% a.out
```

Hacker Guru

```
% cat
Hello, world.
^D
```

Manager începător

```
10 PRINT "HELLO WORLD"
20 END
```

Manager expert

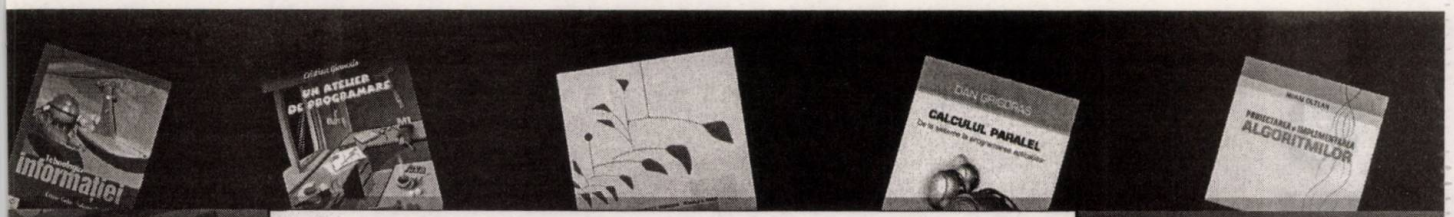
```
mail -s "Hello, world." bob@b12
Bob, dacă te rog frumos, ai putea să-mi faci
un program care tipărește "Hello, world."?
Am nevoie de el mâine.
^D
```

Manager principal

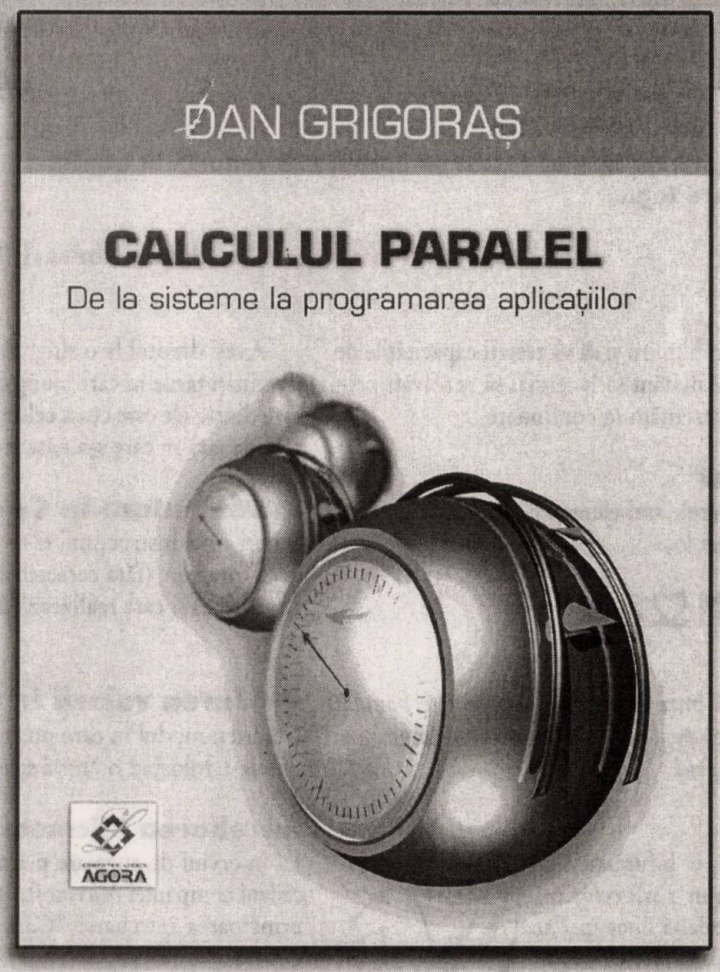
```
% zmail jim
Am nevoie de un program "Hello, world." în
după-amiza asta.
```

Director

```
% letter
letter: Command not found.
% mail
To: ^X ^F ^C
% help mail
help: Command not found.
% la naiba!
!: Event unrecognized
% logout
```



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate, dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Probleme de LOGICĂ

Mihai Scorţaru

"- Când logica nu îți mai folosește trebuie să renunți la logică!
- Mi se pare logic!"

Frank Herbert, Dune

Dacă vreți să vă distrați puțin și să vă testați capacitățile de a raționa "logic", vă invităm să încercați să rezolvați problemele pe care le prezentăm în continuare.

Șir "nerecurent"

Determinați următoarele trei elemente ale șirului prezentat în imaginea de mai jos:

Indicație

Relația de recurență între elementele șirului este banală! Nu aveți nevoie de ea pentru determinarea logicii pe baza căreia este construit șirul.

Iepurașul

Pe un drum există o bifurcație; una dintre variante duce în *Țara celor care mint tot timpul* și cealaltă duce în *Țara celor care spun adevărul întotdeauna.*

Un iepuraș vine dintr-una dintre cele două țări (nu se știe din care) și a trecut deja de bifurcație în momentul în care îl întâlniți. Se știe că iepurașul s-a născut într-una dintre cele două țări, dar nu se știe în care. În plus, nu se știe dacă iepurașul vine din țara sa natală sau din cealaltă.

Iepurașul vă va răspunde la o singură întrebare. Va trebui să alegeți întrebarea astfel încât să determinați care dintre drumuri duce în *Țara celor care mint tot timpul* și care duce în *Țara celor care spun adevărul întotdeauna.*

Dacă iepurașul s-a născut în *Țara celor care mint tot timpul*, atunci el va minți când va răspunde la întrebare, iar dacă s-a născut în *Țara celor care spun adevărul întotdeauna*, atunci va da un răspuns adevărat.

Aveți dreptul la o singură întrebare și va trebui să determinați țările în care ajung cele două drumuri indiferent care dintre ele este cea a celor care spun adevărul și indiferent de țara în care s-a născut iepurașul.

Incrementare în C++

Scrieți cinci instrucțiuni C++ diferite care conțin cel mult cinci caractere (fără caracterul `';'` care urmează după instrucțiune) și care realizează incrementarea cu `1` a variabilei `i`.

Dublarea valorii în C++

Descrieți modul în care poate fi dublată valoarea unei variabile `i` folosind o "navă spațială".

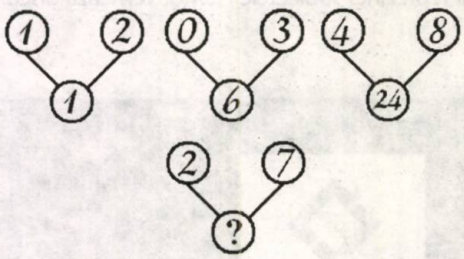
Întrebarea Microsoft

La interviul de angajare pentru funcția de pogramator în cadrul companiei *Microsoft*, "pretendenților" li s-a adresat următoarea întrebare: "*Câte benzinării există în Statele Unite?*"

Încercați să vă imaginați răspunsul la o astfel de întrebare!

Căsuța liberă

Încercați să determinați numărul care ar trebui scris în cerculețul care conține semnul de întrebare!



PC MAGAZINE ROMÂNIA

Concurs de pagini web pentru licee și facultăți
dec. 2002 - ian. 2003

ediția a doua

webdesigner

pentru școala ta

www.pcmagazine.ro/concurs

Sponsor principal:

IBM

prin Partenerul Autorizat

PC-uri IBM Netvista A40i pentru echipa #1!

sponsori

ROMSYS

providing the future rds

THE FIRST 5 YEARS OF EVOLUTION

THE DOCUMENT COMPANY XEROX

Publicarea acestei reviste este posibilă datorită sponsorizărilor acordate de următoarele firme care s-au decis să sprijine învățământul românesc de informatică

SPONSORI DE PLATINĂ

COMPAQ

SPONSORI DE AUR

FORTE COMPUTERS
Flexibility. Made in Romania

ASTRAL TELECOM

SPONSORI DE ARGINT

s&t
IT SOLUTIONS & SERVICES

hp
invent

ERICSSON

Maguay

M.G.T.
EDUCATIONAL

a&C INTERNATIONAL S.A.
the CAD-CAMpany
autodesk
authorized distributor

ALCATEL

*Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

gazeta de informatică
Revista viitorilor specialiști în tehnologia informației
Februarie 2003
Vol. 13/2
Lei 30.000

din sumarul numărului:

focus

LABIRINTURI

babel

Conceptul de algoritm o abordare modernă

Plus

Faza locală a olimpiadei de informatică - Tg. Jiu și Cluj
internet: Pagini Web cu PHP 4 • Teoreme de programare
2 probleme de la ONI '02 • Compresia datelor-episodul 2
enjoy: Probleme de logică • Concurs: rundele nr. 9-12

*a se consuma
înainte de pornirea
calculatorului

Vol. 13, Nr. 2 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. *Clara Ionescu*
Redactor-șef:
Mihai Scorțaru
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Claudiu Soroiu
Au mai colaborat:
Mihai Pătrașcu (Boston - S.U.A.)
conf. dr. *Marin Vlada (București)*
prof. *Radu Vișinescu (Ploiești)*
Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/192.422
0265/166.516
Fax: 0265/166.290
http://www.ginfo.ro
ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 0265/162.359
Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*", "*NetReport*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

probleme.. PAG. 5

Faza LOCALĂ a olimpiadei de INFORMATICĂ

~ enunțurile problemelor ~
~ propuse spre rezolvare ~
~ la Târgu Jiu și la Cluj ~

soluții.. PAG. 13

Despre două PROBLEME de la ONI 2002: noi SOLUȚII și EXTINDERI

Mihai PĂTRAȘCU

concurs.. PAG. 15

Despre rundele #09-#12
Clasament

focus.. PAG. 18

Algoritmi de TARE parcurgere

Radu VIȘINESCU

internet.. PAG. 22

Pagini WEB cu PHP 4

Mihai SCORȚARU
Claudiu SOROIU

babel.. PAG. 35

Conceptul de ALGORITM - abordare MODERNĂ

Marin VLADA

~ despre algoritmi ~
~ prima parte ~

serial.. PAG. 40

COMPRESIA datelor

Claudiu SOROIU

~ al doilea episod ~

begin.. PAG. 45

TEOREME de programare

Clara IONESCU

~ teorie și practică ~

enjoy.. PAG. 48

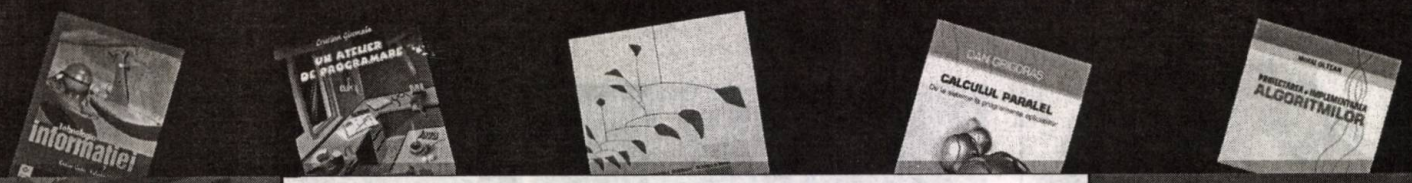
Probleme de LOGICĂ

Mihai SCORȚARU

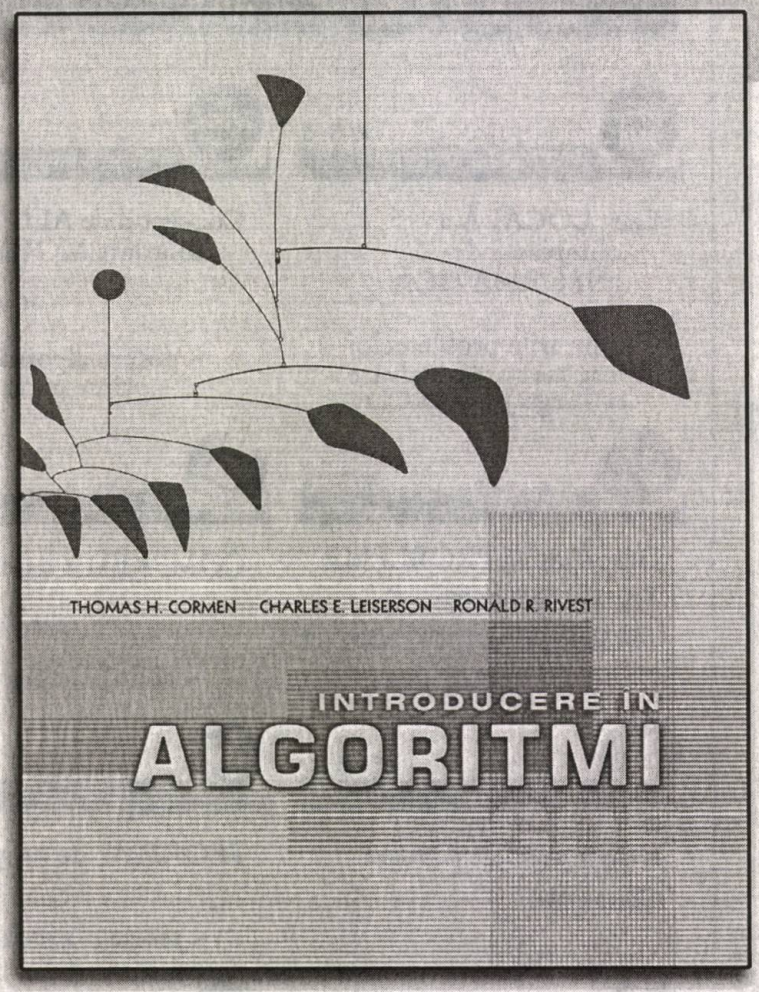
```
program Olimpiada;
...
begin
  Participa;
  Rezolva_Problemele;
  Aplaudi;
end.
```

Redacția GInfo vă urează mult succes la concursurile de programare din acest an!

100 puncte



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Despre ROBOȚEI

Cred că foarte puțini dintre voi nu au auzit de *Isaac Asimov*, scriitor considerat a fi unul dintre cei mai importanți creatori de literatură științifico-fantastică.

După cum probabil știți, o mare parte a romanelor și nuvelelor sale au ca subiect roboții. El consideră că roboții sunt entități raționale care funcționează pe baza unor legi fundamentale.

Funcționarea creierelor pozitronice ale roboților lui *Asimov* se bazează pe trei legi fundamentale, numite ***legile roboticii***:

Prima lege a roboticii: *Un robot nu poate răni nici o ființă umană sau, prin neintervenție, să permită rănirea unei ființe umane.*

A doua lege a roboticii: *Un robot trebuie să îndeplinească ordinele unei ființe umane, atât timp cât aceste ordine nu contravin primei legi.*

A treia lege a roboticii: *Un robot trebuie să își asigure supraviețuirea atâta timp cât aceasta nu contravine primelor două legi.*

Pe parcursul scrierii operelor sale, *Asimov* a efectuat câteva modificări cum ar fi introducerea unor ponderi pentru fiecare lege și chiar introducerea unei noi legi și anume:

Legea Zero a roboticii: *Un robot trebuie să acționeze având ca principal scop supraviețuirea societății umane.*

În condițiile acestei legi prima lege a roboticii devine: *Un robot nu poate răni nici o ființă umană sau, prin neintervenție, să permită rănirea unei ființe umane, atât timp cât aceasta nu contravine legii Zero.*

Evident, legea a doua și legea a treia trebuie modificate corespunzător.

După moartea lui *Asimov* opera sa a fost continuată de diverși scriitori *SF*. Astfel, în condițiile în care tehnologia s-a dezvoltat în așa fel încât roboții se puteau reproduce a apărut legea a patra a roboticii: *Un robot trebuie să acționeze în vederea "perpetuării speciei sale" atât timp cât aceasta nu contravine legilor anterioare.*

În momentul în care în universul operei lui *Asimov* a apărut o specie inteligentă non-umană a fost propusă ***legea Minus Unu a roboticii***: *Un robot trebuie să acționeze având ca principal scop supraviețuirea speciilor inteligente din Univers.* Următoarele legi au fost modificate corespunzător.

Se știe că multe dintre roadele imaginației unor scriitori de *science-fiction* reprezintă baza unor descoperiri științifice reale. Cel mai indicat domeniu în care ar trebui aplicate legi asemănătoare cu cele ale lui *Asimov* este inteligența artificială.

Recent a fost introdus conceptul de agent filosofic. Acesta este un agent inteligent care funcționează pe baza unor legi elementare, numite ***principii filosofice***.

Legile lui *Asimov* au fost transformate astfel încât să modeleze comportamentul agenților atunci când trebuie să îndeplinească o misiune. Cele șapte principii la care s-a ajuns sunt:

Primul principiu al agenților filosofici: *Un agent nu poate împiedica buna desfășurare a misiunii prin acțiunile sale sau prin inacțiune.*

Al doilea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primului principiu, un agent nu va provoca daune celorlalți participanți la misiune.*

Al treilea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primelor două principii, un agent nu își va provoca daune.*

Al patrulea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primelor trei principii, un agent va acționa pentru a atinge scopul misiunii.*

Al cincilea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primelor patru principii, un agent va respecta toate convențiile stabilite.*

Al șaselea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primelor cinci principii, un agent va acționa pentru a-și atinge propriile scopuri.*

Al șaptelea principiu al agenților filosofici: *Cu excepția cazurilor în care aceasta contravine primelor șase principii, un agent va opera eficient.*

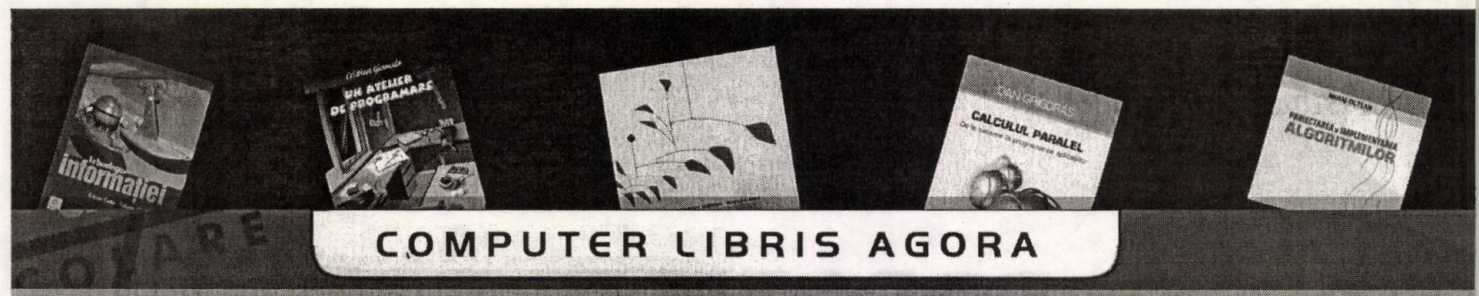
Cele șapte principii pot părea oarecum ciudate, iar ordinea ar putea părea incorectă (de exemplu, eficiența este pusă pe ultimul loc).

Evoluția acestor legi a fost similară cu cea a legilor lui *Asimov*, aducându-se completări și efectuându-se modificări pe parcurs.

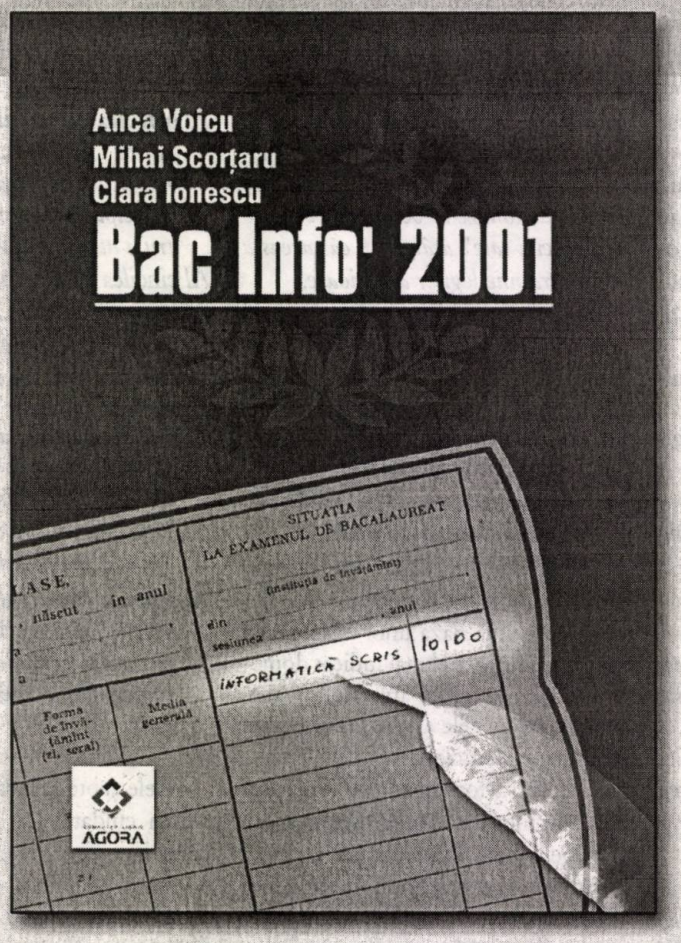
Totuși, cel mai important este faptul că anumite aspecte sunt preluate din literatura de anticipație și adaptate pentru a satisface nevoile actuale ale științei.

Așadar, cred că se poate spune că în viitorul mai mult sau mai puțin îndepărtate, un *R. Daneel* (cel mai celebru robot creat de *Asimov*) real va funcționa pe baza unor principii foarte asemănătoare cu cele pe care le-a descris autorul versiunii "imaginare".

Mihai Scorțaru



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Târgu-Jiu, județul Gorj

Faza LOCALĂ a olimpiadei de INFORMATICĂ

Vă prezentăm în continuare enunțurile problemelor propuse spre rezolvare la ediția 2003 a fazei locale a olimpiadei de informatică de la Colegiul Național "Ecaterina Teodoroiu" din Târgu-Jiu.

P020301: Melodii

clasa a IX-a, prof. Carmen Șelaru

Se consideră o matrice *A* cu patru linii și cinci coloane ale cărei elemente sunt fie zerouri, fie numere mai mari decât 1. Fiecare linie va conține un element cu valoarea 0 și patru elemente care au alte valori.

Liniile sunt numerotate începând cu 1; pe liniile impare, elementul cu valoarea 0 se află pe ultima coloană, iar pe celelalte linii, elementul cu valoarea 0 se află pe prima coloană. Pe liniile impare, celelalte valori sunt cele mai mici patru numere care nu au fost folosite pe liniile anterioare, ordonate crescător de la stânga la dreapta. Pe liniile pare, celelalte valori sunt cele mai mici patru numere care nu au fost folosite pe liniile anterioare, ordonate crescător de la dreapta la stânga. Pe prima linie se vor afla numerele 2, 3, 4 și 5.

Așadar, elementele matricei sunt:

```
2 3 4 5 0
0 9 8 7 6
10 11 12 13 0
0 17 16 15 14
```

Se consideră opt melodii care trebuie înregistrate pe un *CD*. Pentru fiecare se cunoaște durata exprimată în minute și secunde. Numărul de minute este dat de un element de pe o coloană pară a matricei, iar numărul de secunde este reprezentat de un element de pe o coloană impară a acesteia. Așadar, prima melodie durează 3 minute și 2 secunde, a doua 5 minute și 4 secunde, a treia 7 minute și 6 secunde și așa mai departe până la a opta melodie care durează 17 minute și 16 secunde.

Durata totală a înregistrărilor de pe *CD* este de `ttm` minute și `tts` secunde. Între două melodii consecutive de pe *CD* trebuie să existe o pauză de cinci secunde.

Va trebui să alegeți melodiile care vor fi înregistrate astfel încât *CD*-ul să conțină cât mai multe melodii. Melodiile trebuie înregistrate integral.

Date de intrare

Fișierul de intrare **CD.IN** conține două numere întregi care reprezintă valorile `ttm` și `tts`, separate printr-un singur spațiu.

Date de ieșire

Primele patru linii ale fișierului de ieșire **CD.OUT** vor conține câte cinci numere separate prin câte un spațiu. Aceste numere vor reprezenta elementele matricei *A*.

Următoarea linie va conține durata totală a celor opt melodii, exprimată în minute și secunde. Aceste două numere vor fi separate printr-un spațiu.

Cea de-a șasea linie a fișierului va conține numerele de ordine ale melodiilor care au fost alese pentru a fi înregistrate, separate prin câte un spațiu.

Ultima linie a fișierului va conține două numere separate printr-un spațiu, care vor reprezenta spațiul liber rămas pe *CD* după înregistrare, exprimat în minute și secunde.

Exemplu

CD.IN

```
50 4
```

CD.OUT

```
2 3 4 5 0
0 9 8 7 6
10 11 12 13 0
0 17 16 15 14
81 12
0 57
```

P020302: Concurs de șah

clasa a IX-a, prof. Florin Băltărețu

La un concurs participă *n* șahiști; ei sunt împărțiți în mai multe grupuri după criterii valorice. Șahiștii joacă în total *p* partide.

Fiecare jucător al unui grup joacă o partidă cu toți ceilalți din grupul respectiv.

Cunoscându-se pentru fiecare partidă cei doi jucători, să se determine în câte grupuri au fost împărțiți șahiștii și care sunt membri grupului sau grupurilor de dimensiune maximă.

Date de intrare

Prima linie a fișierului de intrare **SAH.IN** va conține numărul n al șahiștilor și numărul p al partidelor; aceste numere vor fi separate printr-un singur spațiu. Fiecare dintre următoarele p linii va conține două numere x și y cu semnificația: *are loc o partidă între jucătorul x și jucătorul y.*

Date de ieșire

Prima linie a fișierului de ieșire **SAH.OUT** va conține numărul grupurilor în care au fost împărțiți șahiștii. Următoarea linie va conține numerele de ordine ale șahiștilor din grupul sau grupurile de dimensiune maximă.

Exemplu

SAH.IN

```
7 9
1 2
2 3
1 3
4 5
5 6
6 7
7 4
6 4
5 7
```

SAH.OUT

```
2
4 5 6 7
```

P020303: Cifre binare

clasa a X-a, prof. Mihaela Runceanu

Se consideră un număr a cuprins între 1 și 100 a cărui reprezentare binară conține m cifre 1 și n cifre 0. Să se determine toate numerele a căror reprezentare binară are exact m cifre 1 și n cifre 0. Reprezentările binare ale acestor numere nu pot conține zerouri nesemnificative.

Date de intrare

Fișierul de intrare **BINAR.IN** va conține o singură linie pe care se va afla valoarea a.

Date de ieșire

Fișierul de ieșire **BINAR.OUT** va conține, pe câte o linie, date referitoare la toate numerele a căror reprezentare binară au același număr de cifre 1 și același număr de cifre 0 ca și numărul a. Pentru fiecare număr va fi afișată valoarea în baza 10, urmată de valoarea în baza 2. Cele două valori vor fi separate prin câte un spațiu.

Exemplu

BINAR.IN

```
65
```

BINAR.OUT

```
65 1000001
66 1000010
68 1000100
72 1001000
80 1010000
96 1100000
```

P020304: Acvariu

clasa a X-a, prof. Dumitru Tomulescu

O persoană are la dispoziție o sumă de bani S cu care dorește să cumpere cât mai mulți pești pentru acvariul pe care l-a primit cu ocazia zilei de naștere. Din nefericie, unele specii de pești nu pot coexista în același acvariu.

Se cunoaște numărul N al speciilor de pești, precum și prețurile p_i (i =1, ..., N) ale peștilor din fiecare specie. De asemenea, se cunosc perechile de specii care nu pot coexista în acvariu. Speciile sunt identificate prin numere cuprinse între 1 și N.

Să se determine numărul maxim de pești care se pot cumpăra folosind integral suma S, știind că:

- trebuie cumpărați pești din cel puțin trei specii;
- pot fi cumpărați oricâți pești din aceeași specie;
- nu pot fi cumpărați pești din două specii care nu pot coexista în acvariu.

Dacă există mai multe soluții trebuie determinată doar una dintre ele.

Date de intrare

Prima linie a fișierului de intrare **ACVARIU.IN** va conține suma S care poate fi folosită pentru cumpărarea peștilor.

Cea de-a doua linie va conține numărul N al speciilor disponibile.

Fiecare dintre următoarele N linii va conține câte două numere naturale i și p_i cu semnficiația: costul unui pește din specia i este p_i.

Pe următoarele linii (până la sfârșitul fișierului) se vor afla câte două numere x și y cu semnificația: *peștii din speciile x și y nu pot coexista în acvariu.*

Date de ieșire

Prima linie a fișierului de ieșire **ACVARIU.OUT** va conține numărul maxim M al peștilor care pot fi achiziționați folosindu-se integral suma S.

Următoarea linie a fișierului va conține M valori întregi reprezentând numerele de ordine ale speciilor din care fac parte cei M pești.

În cazul în care nu există nici o posibilitate de a achiziționa cel puțin trei specii de pești care pot coexista în acvariu și pentru achiziționarea lor se folosește integral suma S, atunci fișierul de ieșire va conține o singură linie pe care se va afla mesajul IMPOSIBIL.

Exemplu

ACVARIU.IN

```
80
4
1 20
2 10
3 15
4 25
1 4
```

ACVARIU.OUT

```
6
1 3 3 2 2 2
```

P020305: Turism

clasele a XI-a și a XII-a, prof. Cristian Șovăilă

Se consideră mai multe orașe între care există drumuri pe care se poate circula în ambele sensuri. Numărul obiectivelor turistice dintr-un oraș este egal cu numărul caracterelor din denumirea orașului.

Un turist dorește să viziteze cât mai multe obiective turistice, eventual trecând de mai multe ori prin același oraș. Chiar dacă trece de mai multe ori printr-un oraș, obiectivele din acel oraș sunt numărate o singură dată (la prima vizită).

Vă trebui să construiți un graf în care nodurile reprezintă orașe, iar muchiile reprezintă drumuri și să determinați matricea de adiacență a acestuia.

De asemenea, va trebui să determinați numărul maxim de obiective pe care le poate vizita turistul știind că poate porni din orice oraș, dar nu poate circula decât pe drumuri (eventual, poate vizita un oraș de mai multe ori), precum și care sunt denumirile orașelor pe care le vizitează.

Date de intrare

Fișierul de intrare **TURISM.IN** va conține mai multe linii, pe fiecare aflându-se denumirile a două orașe, separate prin câte un spațiu; între cele două orașe există un drum pe care se poate circula în ambele sensuri.

Date de ieșire

Primele linii ale fișierului de ieșire **TURISM.OUT** vor conține matricea de adiacență corespunzătoare grafului. Aceasta va conține câte o linie pentru fiecare oraș și fiecare linie va conține un număr de elemente egal cu numărul orașelor. Ordinea în care vor fi scrise liniile corespunde ordinii în care apar orașele în fișierul de intrare. Cel de-al j-lea element de pe a i-a linie va avea valoarea 1 dacă există un drum între orașele corespunzătoare liniilor i și j ale matricei și 0 în caz contrar. Numerele de pe o linie vor fi separate prin câte un spațiu.

Următoarea linie a fișierului va conține numărul maxim de obiective turistice care pot fi vizitate.

Ultima linie a fișierului va conține denumirile orașelor vizitate, separate prin spații.

Exemplu

TURISM.IN

```
craiova severin
vilcea slatina
piatra_neamt iasi
galati iasi
tg-jiu craiova
```

TURISM.OUT

```
0 1 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 1 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
22
piatra_neamt iasi galati
```

P020306: Numere

clasele a XI-a și a XII-a, prof. Vasilica Tomulescu

Se consideră un număr natural N cuprins între 1 și 16. Va trebui să construiți un șir care conține toate numerele cuprinse între 1 și 2^N; acest șir este construit astfel:

- se pornește cu șirul care conține numerele 1 și 2;
- exact la mijlocul acestui șir sunt inserate numerele 3 și 4, obținându-se șirul 1, 3, 4, 2;
- la mijlocul noului șir sunt inserate numerele 5, 6, 7 și 8, obținându-se șirul 1, 3, 5, 6, 7, 8, 4, 2;
- procedeul continuă până la inserarea tuturor celor 2^N numere.

Așadar, la fiecare pas i șirul va conține primele 2^i numere și la mijlocul său vor fi inserate următoarele 2^i numere.

Date de intrare

Fișierul de intrare **NUMERE.IN** va conține o singură linie pe care se va afla valoarea N.

Date de ieșire

Fișierul de ieșire **NUMERE.OUT** va conține o singură linie pe care se vor afla cele 2^N elemente ale șirului construit potrivit regulilor prezentate anterior. Elementele șirului vor fi separate prin câte un spațiu.

Exemplu

NUMERE.IN

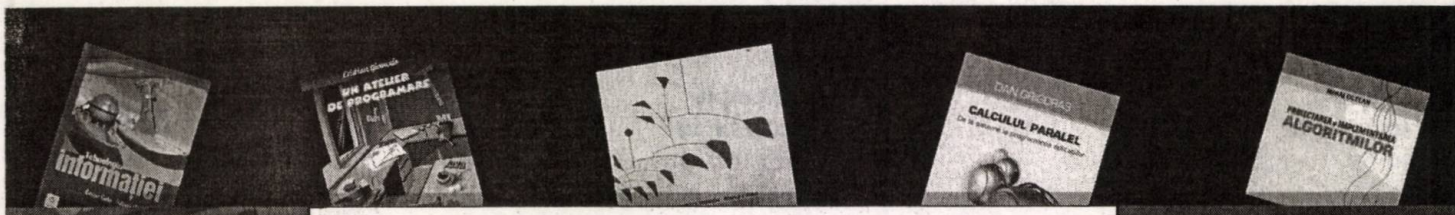
```
4
```

NUMERE.OUT

```
1 3 5 6 9 10 11 12 13 14 15 16 7 8 4 2
```

Observație

Enunțurile au fost modificate de redacția *GInfo* pentru a elimina unele ambiguități și pentru ca problemele să respecte regulile impuse de revista noastră.



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Cluj, 1 februarie 2003

Faza LOCALĂ a olimpiadei de INFORMATICĂ

În acest an, problemele propuse spre rezolvare la acest concurs, în județul Cluj au fost alese în colaborare cu redacția GInfo. Vă prezentăm în continuare enunțurile celor nouă probleme cu care s-au confruntat participanții.

P020307: Ecuație

Se consideră ecuația de forma $x^2 - y^2 = k$. Cunoscându-se valoarea întreagă nenulă k, să se determine toate perechile de numere întregi (x, y) pentru care ecuația anterioară este satisfăcută.

Date de intrare

Fișierul de intrare **ECUATIE.IN** va conține o singură linie pe care se va afla un număr întreg care reprezintă valoarea k.

Date de ieșire

În fișierul de ieșire **ECUATIE.OUT** vor fi scrise toate perechile (x, y), câte una pe o linie. Cele două numere de pe aceeași linie vor fi separate printr-un singur spațiu. În cazul în care nu există nici o astfel de pereche, fișierul de ieșire va conține o singură linie pe care va fi scris mesajul `NU EXISTA SOLUTII`.

Restricții și precizări

- $-1000 < k < 1000$, $k \neq 0$;
- nu contează ordinea în care sunt scrise perechile în fișierul de ieșire;
- o pereche poate apărea în fișierul de ieșire o singură dată.

Exemple

ECUATIE.IN
```
8
```

ECUATIE.OUT
```
3 1
-3 1
-3 -1
3 -1
```

ECUATIE.IN
```
1
```

ECUATIE.OUT
```
1 0
-1 0
```

Timp de execuție: 1 secundă/test

P020308: Număr

Dându-se un număr natural N, să se determine cel mai mic număr M care este par și este format din cifrele numărului N.

Date de intrare

Fișierul de intrare **NUMAR.IN** conține o singură linie pe care se află un număr întreg care reprezintă valoarea N.

Date de ieșire

Fișierul de ieșire **NUMAR.OUT** trebuie să conțină o singură linie pe care se va afla valoarea M. În cazul în care nu există nici un număr par format din cifrele numărului N, pe linia din fișierul de ieșire se va scrie mesajul `NU EXISTA`.

Restricții și precizări

- $1 \leq N \leq 1.000.000.000.000$ (o mie de miliarde);
- numărul M nu poate conține zerouri nesemnificative.

Exemple

NUMAR.IN
```
3216756
```

NUMAR.OUT
```
1235676
```

NUMAR.IN
```
45098060
```

NUMAR.OUT
```
40005698
```

NUMAR.IN
```
3333333333
```

NUMAR.OUT
```
NU EXISTA
```

Timp de execuție: 1 secundă/test

P020309: Sumă

Se consideră un număr $N = \overline{a_1 a_2 \text{L } a_n}$ (cifrele numărului sunt a_1, a_2, ..., a_n). Determinați valoarea sumei:

$$S = \overline{a_1 a_2 \text{L } a_n} + \overline{a_2 a_3 \text{L } a_n a_1} + \text{L} + \overline{a_n a_1 \text{L } a_{n-1}}.$$

Date de intrare

Fișierul de intrare **SUMA.IN** conține o singură linie pe care se află un număr întreg care reprezintă valoarea N.

Date de ieșire

Fișierul de ieșire **SUMA.OUT** trebuie să conțină o singură linie pe care se va afla valoarea S.

Restricții și precizări

- $1 \le N \le 100.000.000$ (o sută de milioane);
- termenii sumei pot conține zerouri nesemnificative;
- numărul N nu conține zerouri nesemnificative.

Exemplu

SUMA.IN
```
1234
```

SUMA.OUT
```
11110
```

Timp de execuție: 1 secundă/test

P020310: Numere

Se consideră un număr natural N cuprins între 1 și 16. Va trebui să construiți un șir care conține toate numerele cuprinse între 1 și 2^N; acest șir este construit astfel:

- se pornește cu șirul care conține numerele 1 și 2;
- înaintea acestui șir este inserat numărul 3, iar la sfârșitul șirului este inserat numărul 4, obținându-se șirul 3, 1, 2, 4;
- înaintea acestui șir sunt inserate numerele 5 și 6 iar la sfârșitul șirului sunt inserate numerele 7 și 8, obținându-se șirul 5, 6, 3, 1, 2, 4, 7, 8;
- procedeul continuă până la inserarea tuturor celor 2^N numere.

Așadar, la fiecare pas i șirul va conține primele 2^i numere și înaintea sa vor fi inserate următoarele 2^{i-1} numere, iar la sfârșitul său vor fi inserate următoarele 2^{i-1} numere.

Date de intrare

Fișierul de intrare **NUMERE.IN** va conține o singură linie pe care se va afla valoarea N.

Date de ieșire

Fișierul de ieșire **NUMERE.OUT** va conține o singură linie pe care se vor afla cele 2^N elemente ale șirului construit potrivit regulilor prezentate anterior. Elementele șirului vor fi separate prin câte un spațiu.

Exemplu

NUMERE.IN
```
4
```

NUMERE.OUT
```
9 10 11 12 5 6 3 1 2 4 7 8 13 14 15 16
```

Timp de execuție: 1 secundă/test

P020311: Triunghi

Se consideră un triunghi (determinat de trei puncte necoliniare din plan) și alte N puncte din plan.

Să se determine câte dintre aceste N puncte se află în interiorul triunghiului. Un punct este considerat a fi în interiorul triunghiului și în cazurile în care se află pe una dintre cele trei laturi sau este unul dintre cele trei vârfuri ale triunghiului.

Date de intrare

Primele trei linii ale fișierului de intrare **TRIUNGHI.IN** conțin câte două numere întregi reprezentând coordonatele vârfurilor triunghiului. Următoarea linie conține numărul N al celorlalte puncte. Fiecare dintre următoarele N linii va conține câte două numere întregi reprezentând coordonatele unui punct.

Date de ieșire

Fișierul de ieșire **TRIUNGHI.OUT** trebuie să conțină o singură linie pe care se va afla numărul punctelor din interiorul triunghiului.

Restricții și precizări

- $1 \le N \le 100.000$;
- coordonatele punctelor sunt numere naturale cuprinse între 0 și 255;
- coordonatele tuturor punctelor sunt date în ordinea: coordonata pe axa Ox, coordonata pe axa Oy.

Exemple

TRIUNGHI.IN
```
0 0
10 0
0 10
3
1 1
5 5
10 1
```

TRIUNGHI.OUT
```
2
```

TRIUNGHI.IN

```
0 0
1 0
0 1
3
1 1
2 2
3 3
```

TRIUNGHI.OUT

```
0
```

Timp de execuție: 1 secundă/test

P020312: Găuri

Se consideră o matrice ale cărei elemente fac parte din mulțimea {0, 1}. O *regiune* este o parte a matricei care conține elemente cu aceeași valoare astfel încât se poate ajunge din orice element al regiunii în oricare altul prin deplasări pe orizontală sau verticală, trecând doar prin elemente ale regiunii. O *gaură* este o regiune de zerouri care nu "atinge" marginile matricei. Va trebui să determinați numărul de elemente pe care le conține cea mai mare gaură (cea care conține un număr maxim de elemente).

Date de intrare

Prima linie a fișierului de intrare **GAURI.IN** conține două numere întregi M și N care reprezintă numărul liniilor, respectiv numărul coloanelor matricei. Fiecare dintre următoarele M linii conțin câte N cifre din mulțimea {0, 1}. Acestea reprezintă valorile elementelor matricei. Valorile de pe o linie *nu* vor fi separate prin spații.

Date de ieșire

Fișierul de ieșire **GAURI.OUT** trebuie să conțină o singură linie pe care se va afla numărul elementelor din cea mai mare gaură.

Restricții și precizări

- $1 \le M, N \le 100$;
- dacă nu există nici o gaură, se consideră că cea mai mare gaură are 0 elemente;
- pot exista mai multe găuri care au același număr maxim de elemente.

Exemplu

GAURI.IN

```
8 8
00000000
00100111
01110101
01011011
01001010
01101001
00111111
00000000
```

GAURI.OUT

```
4
```

Timp de execuție: 1 secundă/test

P020313: Exponent

Se consideră două numere întregi m și n. Să se determine cea mai mare valoare k astfel încât numărul $m!$ este multiplu al numărului n^k.

Date de intrare

Fișierul de intrare **EXPONENT.IN** conține o singură linie pe care se află două numere întregi, separate printr-un spațiu, reprezentând valorile m și n.

Date de ieșire

Fișierul de ieșire **EXPONENT.OUT** va conține o singură linie pe care se va afla valoarea k.

Restricție

- $2 \le m, n \le 2.000.000.000$ (două miliarde).

Exemplu

EXPONENT.IN

```
20 24
```

EXPONENT.OUT

```
6
```

Timp de execuție: 1 secundă/test

P020314: Soldați

Se consideră n soldați care fac parte dintr-un detașament și n valori care reprezintă numărul de pachete de hrană pe care le poate duce fiecare dintre soldați (prima valoare corespunde primului soldat, a doua valoare celui de-al doilea soldat etc.). Soldații trebuie dispuși într-un număr nenul b de bastioane, respectându-se condițiile:

- nu pot exista bastioane fără soldați;
- bastioanele sunt identificate prin numere cuprinse între 1 și b și pot fi distruse, dar numai unul după altul, începând cu primul (cu alte cuvinte, un bastion nu poate fi distrus dacă nu sunt distruse toate bastioanele din fața lui).
- pentru orice număr d de bastioane ($0 \le d \le b - 1$) care pot fi distruse, pachetele de hrană purtate de soldații din bastionul $d + 1$ trebuie să poată fi împărțite exact (fără fracțiuni de pachet) la toți soldații rămași în bastioanele identificate prin numere cuprinse între $d + 1$ și b (cele care nu au fost distruse).

Date de intrare

Prima linie a fișierului de intrare **SOLDATI.IN** va conține numărul n al soldaților. Cea de-a doua linie va conține cele n valori care reprezintă numărul de pachete de hrană pe care le poate duce fiecare soldat. Aceste valori vor fi separate prin câte un spațiu.

Date de ieşire

Fişierul de ieşire **SOLDATI.OUT** va conţine un număr de linii egal cu numărul bastioanelor. Fiecare linie va conţine numărul pachetelor pe care le pot duce soldaţii din bastionul respectiv. Aceste numere vor fi separate prin câte un spaţiu.

Restricţii şi precizări

- $2 \le n \le 1.000$;
- un soldat va duce cel puţin unul şi cel mult 1000 de pachete;
- dacă există mai multe soluţii, trebuie generată doar una dintre ele;
- pentru datele de test folosite va exista întotdeauna cel puţin o soluţie.

Exemplu

SOLDATI.IN

```
6
4 10 3 9 2 3
```

SOLDATI.OUT

```
2 4
3 9
10
3
```

Explicaţie

Detaşamentul este format din şase soldaţi. Soldaţii de la primul bastion pot duce 6 pachete care pot fi împărţite la cei şase soldaţi (cazul în care nu cade nici un bastion).

Soldaţii de la al doilea bastion pot duce 12 pachete care pot fi împărţite la cei patru soldaţi rămaşi (cazul în care nu cade primul bastion).

Soldatul de la al treilea bastion poate duce 10 pachete care pot fi împărţite la cei doi soldaţi rămaşi (cazul în care cad primele două bastioane).

Soldatul de la ultimul bastion poate duce 3 pachete. Acestea nu mai trebuie împărţite deoarece a rămas un singur soldat. ***Dacă la ultimul bastion rămas ar fi fost plasaţi mai mulţi soldaţi, atunci numărul total de pachete pe care le duc soldaţii de la acest bastion trebuie să poată fi împărţit la numărul soldaţilor de la acest bastion.***

Dispunerea tuturor soldaţilor într-un singur bastion nu este posibilă, deoarece numărul total al pachetelor de hrană este 31, care nu este multiplu de 6.

Timp de execuţie: 1 secundă/test

P020315: Graf

Se consideră un graf conex cu n vârfuri şi m muchii. Nodurile sunt identificate prin numere cuprinse între 1 şi n. Să se determine nodul aflat la cea mai mare distanţă faţă de nodul identificat prin numărul 1. Distanţa este dată de lungimea minimă a unui drum de la nodul respectiv la nodul identificat prin numărul 1.

Date de intrare

Prima linie a fişierului de intrare **GRAF.IN** va conţine numărul n al nodurilor şi numărul m al muchiilor, separate printr-un spaţiu. Fiecare dintre următoarele m linii va descrie o muchie a grafului; ea va conţine două numere x şi y, separate printr-un spaţiu, cu semnificaţia: *există o muchie între nodul identificat prin numărul x şi nodul identificat prin numărul y.*

Date de ieşire

Fişierul de ieşire **GRAF.OUT** va conţine o singură linie pe care se va afla numărul prin care este identificat nodul aflat la cea mai mare distanţă faţă de nodul identificat prin numărul 1.

Restricţii şi precizări

- $1 \le n \le 100$;
- $n \le m \le 1000$;
- nu pot exista două sau mai multe muchii între aceleaşi două noduri ale grafului;
- dacă există mai multe noduri aflate la distanţa maximă, poate fi ales oricare dintre ele.

Exemple

GRAF.IN

```
7 8
1 2
1 3
2 3
2 4
3 4
1 5
1 6
1 7
```

GRAF.OUT

```
4
```

GRAF.IN

```
4 4
1 2
1 3
1 4
2 4
```

GRAF.OUT

```
2
```

Timp de execuţie: 1 secundă/test

Observaţie

Evaluarea soluţiilor participanţilor din judeţul *Cluj* a fost centralizată fiind coordonată de redacţia *GInfo*. Pentru evaluare s-a folosit cea mai nouă versiune a sistemul de evaluare creat de redacţia *GInfo*, care a fost utilizat la ediţia din acest an a concursului de programare *Bursele Agora*.

Despre două PROBLEME de la ONI 2002: noi SOLUȚII și EXTINDERI

Mihai Pătrașcu

Acest articol se referă pe scurt la două probleme propuse de autor la Olimpiada Națională de Informatică 2002. Enunțurile acestor probleme au fost publicate în numărul 12/5 (mai 2002) al GInfo. Articolul aduce elemente de noutate, prezentând câte o extindere și o nouă metodă de rezolvare pentru problemele Suma divizorilor și EQS.

Suma divizorilor

În [2] este exprimată opinia că rezolvarea problemei ***Suma divizorilor*** de la ediția 2002 a *Olimpiadei Naționale de Informatică* necesită unele elemente de matematică superioară. Luând în considerare rezolvarea publicată în [1] nu putem decât să fim de acord. Această rezolvare se baza pe existența inversului multiplicativ pentru orice număr nenul într-un corp de resturi modulo 9901 (care este un număr prim). Acest element de matematică ar trebui să fie accesibil elevilor de clasa a XII-a, însă nu și celor de clasa a XI-a (problema a fost propusă la clasele a XI-a și a XII-a).

Când a fost propusă această problemă, a fost luată în vedere și o soluție accesibilă elevilor de clasa a XI-a, care, pe lângă accesibilitate, mai are meritul că nu folosește faptul că 9901 este număr prim (astfel, problema este generalizată pentru numere care sunt resturi modulo numere compuse).

Pentru a prezenta soluția alternativă a acestei probleme, să ne reamintim că determinarea sumei divizorilor numărului A^B a fost redusă la calculul sumei unei progresii geometrice modulo 9901: $S_n = (1 + q + q^2 + ... + q^n) \bmod 9901$ (vezi [1]).

Această sumă se poate calcula eficient dacă observăm următoarea recurență (toate calculele se efectuează modulo 9901):

$$\begin{aligned} S_{2k+1} &= 1 + q + q^2 + ... + q^{2k} + q^{2k+1} = \\ &= 1 + q \cdot (1 + q + q^2 + ... + q^{2k}) = \\ &= 1 + q \cdot S_{2k}. \end{aligned}$$

$$\begin{aligned} S_{2k} &= 1 + q + q^2 + ... + q^k + q^{k+1} + ... + q^{2k} = \\ &= (1 + q + q^2 + ... + q^k) + q^k \cdot (q + q^2 + ... + q^k) = \\ &= S_k + q^k \cdot (S_k - 1). \end{aligned}$$

Dacă folosim o metodă cu timp de execuție logaritmic pentru a calcula valoarea q^k se observă că suma poate fi evaluată într-un timp cu ordinul de complexitate $O(\log^2 N)$. O astfel de metodă pentru calculul valorilor q^k este folosirea unei recurențe similare cu cea de mai sus:

$$q^{2k+1} = q \cdot q^{2k}$$
$$q^{2k} = (q^k)^2.$$

Deși folosirea acestui algoritm cu ordinul de complexitate $O(\log^2 N)$ era suficientă pentru a obține punctajul maxim la concurs, se observă că algoritmul poate fi îmbunătățit pentru a rula într-un timp de ordinul $O(\log N)$ dacă îmbinăm cele două recurențe și calculăm simultan q^n și S_n:

$$q^{2k+1} = q \cdot q^{2k} \qquad S_{2k+1} = 1 + q \cdot S_{2k}$$
$$q^{2k} = (q^k)^2 \qquad S_{2k} = S_k + q^k \cdot (S_k - 1).$$

Înainte de a încheia discuția problemei, dorim să combatem o obiecție pe care am auzit-o referitor la problemă, și anume că unii concurenți puteau să nu cunoască unele proprietăți cum ar fi:

$(A + B) \bmod C = ((A \bmod C) + (B \bmod C)) \bmod C.$

Considerăm că o astfel de obiecție este total nejustificată, având în vedere nivelul înalt de pregătire pe care ar tre-

bui să îl aibă participanții la faza națională a olimpiadei. În primul rând, astfel de proprietăți sunt intuitive și se folosesc încă din clasa a V-a când se calculează ultima cifră a rezultatului unor expresii. În al doilea rând, calculul cu resturi este destul de important în informatică (numere pseudoaleatoare, permutări, *hashing* etc.), astfel încât orice elev cu un interes deosebit pentru informatică (cum ar trebui să fie concurenții de la *ONI*) să îl fi cunoscut și folosit.

EQS

Așa cum a fost propusă, problema *EQS* cere numărarea soluțiilor dintr-un anumit domeniu de forma $[-A, +A]$ a unor ecuații de tipul :

$$a_1 \cdot x^3 + a_2 \cdot y^3 + a_3 \cdot z^3 + a_4 \cdot u^3 + a_5 \cdot v^3 = 0.$$

Pe scurt, soluția consta în a trece două dintre necunoscute în membrul drept al ecuației, schimbând semnul pentru coeficienți. Se generau apoi toate valorile posibile pentru membrul cu două necunoscute și se sortau. În continuare, se generau toate valorile posibile pentru membrul cu trei necunoscute și se căutau logaritmic între valorile posibile ale celuilalt membru al ecuației, folosind o căutare binară. Așadar, se folosește o cantitate de memorie de ordinul $O(A^2)$ (pentru tabloul sortat) și este necesar un timp de execuție de ordinul $O(A^3 \cdot \log A)$ (factorul dominant este căutarea în tabloul sortat).

Propunem în continuare o extensie a acestei probleme pentru ecuații cu șase necunoscute de forma:

$$a_1 \cdot x^3 + a_2 \cdot y^3 + a_3 \cdot z^3 + a_4 \cdot u^3 + a_5 \cdot v^3 + a_6 \cdot v^3 = 0.$$

Menționăm că această extensie s-a aflat în vederea comisiei de la *ONI*, dar nu a fost folosită în concurs, rezolvarea fiind considerată destul de dificilă.

Încercând să folosim soluția de mai sus pentru această nouă problemă, ne lovim de imposibilitatea de a păstra în memorie toate valorile posibile ale unui membru cu trei necunoscute.

Numărul acestor valori poate atinge $100^3 = 1.000.000$; astfel, ar fi necesară o cantitate de memorie disponibilă de 4MB, ceea ce nu a fost cazul la *ONI* 2002.

Intuim fără mare dificultate soluția: dacă am putea să generăm toate valorile posibile ale unui membru cu trei necunoscute în ordine crescătoare, am putea interclasa pur și simplu listele de valori posibile pentru membrul stâng și membrul drept.

Algoritmul ar necesita un spațiu de memorie constant și un timp de execuție constant în plus față de codul de generare a valorilor.

Ca urmare, subproblema care trebuie rezolvată constă în generarea în ordine crescătoare a elementelor următoarei mulțimi:

$$S = \{a \cdot x^3 + b \cdot y^3 + c \cdot z^3 \mid x, y, z \in [-A, A]\}.$$

În primul rând se observă faptul că semnul coeficienților nu contează, deci putem lucra numai cu coeficienți pozitivi.

Într-adevăr, următoarea egalitate este valabilă:

$$\{a \cdot x^3 + b \cdot y^3 + c \cdot z^3 \mid x, y, z \in [-A, A]\} = \{|a| \cdot x^3 + |b| \cdot y^3 + |c| \cdot z^3 \mid x, y, z \in [-A, A]\}.$$

Egalitatea acestor mulțimi se datorează faptului că, dacă este luat în considerare un triplet de forma (x, y, z), atunci vor fi luate în considerare toate tripletele de forma $(\pm x, \pm y, \pm z)$. În cazul în care coeficienții sunt pozitivi, există o relație de ordine parțială foarte utilă între valorile mulțimii și anume:

$$z_1 < z_2 \Rightarrow a \cdot x^3 + b \cdot y^3 + c \cdot z_1^3 \le a \cdot x^3 + b \cdot y^3 + c \cdot z_2^3.$$

Soluția pe care o propunem se bazează pe această relație de ordine parțială. Vom păstra un tablou bidimensional `t` de dimensiuni $2 \cdot A \times 2 \cdot A$ în care vom păstra, pentru fiecare pereche (x, y), cea mai mică valoare z pentru care tripletul (x, y, z) nu a fost încă luat în considerare. Versiunea în pseudocod a algoritmului este următoarea:

se inițializează tabloul `t` *astfel încât* $t_{ij} = -A$
cât timp *există elemente neutilizate ale mulțimii* **execută**
  găsește o pereche `(x, y)` *care minimizează valoarea expresiei* $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$ (*)
  următorul element al mulțimii este $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$
  $t_{xy} \leftarrow t_{xy} + 1$
sfârșit cât timp

Ordinul de complexitate al operației de generare a unui element este dictat de eficiența cu care putem efectua operația marcată cu `(*)` în algoritmul descris anterior. Pentru a executa eficient această operație, vom menține un *heap* în care vor fi păstrate toate perechile (x, y). Acest *heap* va implementa o coadă de priorități, unde prioritatea unui element (x, y) este, bineînțeles, chiar $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$.

Perechea (x, y) "minimă" va fi regăsită în timp constant (este chiar vârful *heap*-ului), iar pentru a menține structura *heap*-ului va mai fi necesar un timp de ordinul $O(\log A^2) = O(2 \cdot \log A) = O(\log A)$, deoarece prioritatea perechii (x, y) se schimbă în momentul incrementării valorii t_{xy}, deci elementul va trebui "cernut" în *heap*.

În concluzie, algoritmul rulează în timp $O(A^3 \cdot \log A)$, și are nevoie de un spațiu de memorie de ordinul $O(A^2)$. În mod suprinzător, aceste limite asimptotice sunt identice cu cele pe care le atingea soluția mai puțin performantă în cazul problemei "ușoare" cu cinci necunoscute.

Bibliografie

1. Mihai Scorțaru, *Olimpiada Națională de Informatică*, *GInfo* 12/7 (noiembrie 2002), p. 8-13, Editura Agora Media, Târgu Mureș
2. Mihai Scorțaru, *După ONI 2002*, *GInfo* 12/8 (decembrie 2002), p. 40-43, Editura Agora Media, Târgu Mureș

Mihai Pătrașcu este student la Massaschussets Institute of Technology (MIT) din Boston (Statele Unite ale Americii) și poate fi contactat prin e-mail la adresa **mip@mit.edu**.

Rundele #09 - #12

Vom prezenta în cadrul acestui articol câteva informații referitoare la desfășurarea a patru runde ale concursului de programare organizat de revista noastră. De asemenea, veți putea vedea clasamentul primilor o sută clasați după desfășurarea primelor 12 runde.

P010304: Piramidă

Există mai multe modalități de rezolvare a acestei probleme, toate necesitând cunoștințe minime de geometrie în spațiu. Pentru fiecare punct dat va trebui să verificăm dacă se află sau nu în interiorul piramidei.

O posibilitate este de a împărți spațiul în subspații determinate de planul în care se află grupuri de trei vârfuri ale piramidei. Dacă punctul considerat și cel de-al patrulea vârf al piramidei se află în semispații diferite, atunci punctul nu se află în interiorul piramidei.

Datorită faptului că un punct este considerat a fi în interiorul piramidei chiar dacă se află pe una dintre fețe, pe una dintre muchii sau este un vârf, vom considera că cele două puncte se află în același semispațiu chiar dacă unul dintre ele se află în planul care delimitează cele două semispații.

Așadar, va trebui să determinăm ecuațiile celor patru plane în care se află cele patru fețe ale piramidei. Un astfel de plan va fi determinat de cele trei puncte care formează fața respectivă.

De asemenea, vom determina valorile ecuațiilor pentru vârful piramidei opus feței corespunzătoare.

În continuare, pentru fiecare punct în parte vom considera pe rând cele patru planuri și vom determina valorile ecuațiilor pentru coordonatele punctului respectiv. Dacă valoarea unei ecuații pentru punctul considerat are semnul diferit de cel al valorii ecuației pentru cel de-al patrulea vârf, atunci cele două puncte se află în semispații diferite, deci punctul se află în exteriorul piramidei.

Pentru a determina ecuația unui plan determinat de trei puncte ale căror coordonate se cunosc, se folosește următoarea relație:

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0,$$

unde x, y, z sunt variabile, iar cele trei puncte care determină planul au coordonatele (x_1, y_1, z_1), (x_2, y_2, z_2), respectiv (x_3, y_3, z_3).

Pentru fiecare punct considerat vom înlocui variabilele x, y și z cu coordonatele punctului și vom determina semnul rezultatului.

O variantă mai simplă de rezolvare se bazează pe faptul că dacă un punct se află în interiorul piramidei, pe una din fețe, pe una din muchii sau este un vârf, atunci suma volumelor celor patru piramide determinate de punctul respectiv și grupuri de trei vârfuri ale piramidei este egală cu volumul piramidei.

Așadar, pentru fiecare punct în parte vom determina volumele piramidelor determinate de punctul respectiv și trei vârfuri ale piramidei. Dacă suma celor patru volume este egală cu volumul piramidei, atunci punctul se află în interiorul piramidei.

Se observă că dacă punctul este un vârf al piramidei, trei dintre cele patru volume vor fi egale cu 0, iar cel de-al patrulea volum va fi chiar volumul piramidei.

În cazul în care vârful se află pe una dintre muchii (dar nu este un vârf al piramidei), atunci două volume vor fi egale cu 0, iar dacă vârful se află pe una dintre fețe (dar nu pe o muchie), atunci unul dintre volume va fi egal cu 0.

Volumul unei piramide pentru care se cunosc coordonatele celor patru vârfuri este dat de valoarea

$$\left| \frac{1}{6} \cdot \begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} \right|.$$

Există multe alte posibilități de rezolvare a acestei probleme folosind diferite elemente de geometrie.

Pentru această problemă am primit 35 de soluții pentru care a fost acordat punctajul maxim. Concurenții care au primit 100 de puncte la cea de-a noua etapă a concursului nostru sunt: *Adrian Balașko (Zalău), Adrian Cârcu (Bistrița), Ágota Turánicz (Cluj-Napoca), Alexandru Sofronia (Bacău), Alexandru Stan (Ploiești), Andrei Csibi (Cluj-Napoca), Andrei Homescu (Târgu Jiu), Andrei Păduraru (Vaslui), Asif-ul Haque (Dhaka - Bangladej), Bogdan Nicolae (București), Ciprian Cană (Vaslui), Constantin Răuțu (Botoșani), Cornel Mihăilă (Brașov), Cristian Bojinovici (Bu-*

cureşti), Csaba Pătcaş (Cluj-Napoca), Dan Ghinea (Bucureşti), Dumitru Codreanu (Chişinău - Republica Moldova), Florin Antonescu (Băicoi), Horea-Alexandru Caramizaru (Craiova), Ionuţ-Bogdan Petre (Plopeni), Iulian Blajut (Bacău), Ivan Kazmenko (Sankt Petersburg - Federaţia Rusă), Marius Nicolae (Slatina), Maximilian Machedon (Bucureşti), Mihail Popa (Bucureşti), Mihail-Cosmin Piţ-Rada (Drobeta Turnu-Severin), Mugurel-Ionuţ Andreica (Bucureşti), Radu Vatavu (Suceava), Răzvan Albu (Bucureşti), Robert-Emanuel Nica (Deleni), Silvestru-Cosmin Negruşeri (Bistriţa), Slotta Johannes (Jena - Germania), Ştefan Ciobâcă (Suceava), Tiberiu Chiorean (Baia Mare) şi Valentin Bran (Satu Mare).

Numărul total al concurenţilor care au trimis soluţii pentru această problemă a fost 66; aşadar, mai mult de jumătate dintre ei au rezolvat-o corect.

P010305: Poarta stelară

Este evident faptul că problema se reduce la determinarea unui ciclu hamiltonian de cost minim într-un graf neorientat. Este cunoscut faptul că această problemă este *NP-completă*. Mai mult, cel mai rapid algoritm cunoscut pentru rezolvarea ei are ordinul de complexitate $O(2^{n/2})$, unde n este numărul nodurilor grafului.

Din aceste motive, pentru această problemă nu putea fi creat un program care să determine cea mai bună soluţie în limita de timp impusă. Ca urmare, la această rundă am avut un nou "concurs de euristici". De data aceasta câştigătorul a fost *Mugurel-Ionuţ Andreica* din *Bucureşti*, care a avut cele mai bune rezultate pentru 14 dintre cele 20 de teste.

Lista primilor 20 de concurenţi clasaţi în "concursul de euristici" de la această rundă este prezentată în continuare:

1. Mugurel-Ionuţ Andreica, Bucureşti	*99*
2. Adrian Cârcu, Bistriţa	*97*
3. Mihail Popa, Bucureşti	*87*
4. Csaba Pătcaş, Cluj-Napoca	*80*
5. Ciprian Cană, Vaslui	*79*
5. Iuliu Konya, Baia Mare	*79*
7. Silvestru-Cosmin Negruşeri, Bistriţa	*78*
8. Tiberiu Chiorean, Baia Mare	*76*
9. Lucian Suditu, Vaslui	*71*
10. Maximilian Machedon, Bucureşti	*69*
11. Dan Ghinea, Bucureşti	*67*
11. Radu Vatavu, Suceava	*67*
13. Ştefan Ciobâcă, Suceava	*65*
14. Dan-Ionuţ Fechete, Suceava	*64*
15. Andrei Homescu, Târgu Jiu	*62*
16. Adrian Balaşko, Zalău	*61*
16. Silviu Hagea, Călăraşi	*61*
18. Slotta Johannes, Jena - Germania	*57*
19. Mircea Pasoi, Ploieşti	*32*
20. Horea Alexandru Caramizaru, Craiova	*31*

La această rundă au fost evaluate soluţiile trimise de 43 dintre concurenţii înscrişi la această ediţie a concursului.

P010306: Ordine

Determinarea celui mai lung şir crescător este una dintre problemele clasice de algoritmică. Este una dintre primele probleme studiate care se rezolvă folosind metoda *programării dinamice*. Totuşi, algoritmul clasic are ordinul de complexitate $O(n^2)$, unde n este numărul elementelor şirului. Ca urmare, timpul necesar executării unui program pentru determinarea celui mai lung subşir crescător cu un milion de elemente este inacceptabil. Aşadar, trebuia folosită o versiune a algoritmului care are ordinul de complexitate $O(n \cdot \log n)$. Această metodă este şi ea destul de cunoscută şi a mai fost prezentată în *GInfo*.

O dovadă în plus în acest sens este numărul mare de participanţi care au trimis toate cele douăzeci de fişiere de ieşire corecte. Cei 31 de concurenţi care au rezolvat corect problema sunt: *Adrian Cârcu (Bistriţa), Andrei Homescu (Târgu Jiu), Andrei Păduraru (Vaslui), Asif-ul Haque (Dhaka - Bangladej), Aurelian Marinescu (Braşov), Bogdan Nicolae (Bucureşti), Ciprian Cană (Vaslui), Constantin Jucovschi (Chişinău - Republica Moldova), Cornel Mihăilă (Braşov), Cristian Bojinovici (Bucureşti), Csaba Pătcaş (Cluj-Napoca), Dan Ghinea (Bucureşti), Daniel Nechifor (Vaslui), Dan-Ionuţ Fechete (Suceava), Dumitru Codreanu (Chişinău - Republica Moldova), Gabriel-Alexandru Rudeanu (Iaşi), Iulian Blajut (Bacău), Iuliu Konya (Baia Mare), Ivan Kazmenko (Sankt Petersburg - Federaţia Rusă), Lego Haryanto (Petaluma - Statele Unite ale Americii), Lucian Suditu (Vaslui), Maximilian Machedon (Bucureşti), Mihail Popa (Bucureşti), Mircea Pasoi (Ploieşti), Mugurel-Ionuţ Andreica (Bucureşti), Răzvan Albu (Bucureşti), Răzvan Tigaeru (Călăraşi), Robert-Emanuel Nica (Deleni), Silvestru-Cosmin Negruşeri (Bistriţa), Ştefan Ciobâcă (Suceava)* şi *Tiberiu Chiorean (Baia Mare)*.

Numărul concurenţilor care au trimis fişiere de ieşire pentru această problemă a fost: 68.

P010307: Sortare

Problema cere sortarea unui şir de numere folosind cât mai puţine comparaţii între elementele acestuia.

Din punct de vedere al numărului de comparaţii efectuate, cel mai performant algoritm de sortare este *merge-insert*. Acesta poate fi găsit în volumul III al cărţii *Arta programării calculatoarelor*, scrisă de *Donald Knuth*.

O sortare prin inserţie folosind căutarea binară necesită un număr de $\sum_{i=1}^{n} \lceil \log i \rceil$ comparaţii. Folosind această metodă de sortare se puteau obţine 79 dintre cele 100 de puncte. Dacă se utilizează algoritmul *merge-insert*, atunci numărul de comparaţii se reduce la $\sum_{i=1}^{n} \left\lceil \frac{3}{4} \cdot \log i \right\rceil$.

Doar şapte dintre cei 52 de concurenţi care au trimis soluţii au obţinut punctajul maxim. Aceştia sunt: *Adrian Cârcu (Bistriţa), Adrian Nasui (Slobozia), Andrei-Gabriel Ion (Constanţa), Iuliu Konya (Baia Mare), Maximilian Machedon (Bucureşti), Mugurel-Ionuţ Andreica (Bucureşti)* şi *Silvestru-Cosmin Negruşeri (Bistriţa)*.

Clasament după 12 runde

1. Adrian Cârcu, Bistrița	1171
2. Mugurel-Ionuț Andreica, București	1103
3. Ivan Kazmenko, Sankt Petersburg - Federația Rusă	1028
4. Silvestru-Cosmin Negrușeri, Bistrița	965
5. Csaba Pătcaș, Cluj-Napoca	961
6. Dan Ghinea, București	943
7. Aurelian Marinescu, Brașov	929
8. Dumitru Codreanu, Chișinău - Republica Moldova	920
9. Ciprian Cană, Vaslui	899
10. Ștefan Ciobâcă, Suceava	897

11. Radu Vatavu, Suceava	875	40. M. Piț-Rada, Dr. Tr. Severin	364	69. Costel Antohi, Tecuci	173
12. Bogdan Nicolae, București	872	41. Radu Ștefan, Brașov	352	70. Ralph Bottesch, Sibiu	166
13. Cornel Mihăilă, Brașov	817	42. Mohamed Taha, Egipt	350	71. Alex. Constantin, Ploiești	164
14. Andrei Homescu, Târgu Jiu	808	43. Bogdan Pănuș, Satu Mare	334	72. Saratov Team, Rusia	155
15. Dan-Ionuț Fechete, Suceava	755	44. Ștefăniță Fechete, Suceava	333	72. Ștefan Bucur, București	155
16. Daniel Nechifor, Vaslui	751	45. Marius Airinei, Bacău	324	74. Natalia Baranova, Rusia	154
16. Ionuț-Bogdan Petre, Plopeni	751	45. Radu Berinde, București	324	75. G. Matei, Câmpulung Muscel	147
18. Tiberiu Chiorean, Baia Mare	749	47. Călin Bindea, Bistrița	307	76. Mihaela Marin, Codlea	146
19. Iulian Blajut, Bacău	745	48. Lego Haryanto, SUA	306	77. Daniel Păsăilă, Suceava	144
20. Iuliu Konya, Baia Mare	738	49. Robert Nica, Deleni	305	78. Bogdan Chirițoiu, Ploiești	143
21. Mihail Popa, București	711	50. Ágota Turánicz, Cluj-Napoca	287	79. Alex. Vlajoaga, Pitești	137
22. M. Machedon, București	682	51. Valentin Bran, Satu Mare	277	79. Vincent Groenhuis, Olanda	137
23. Marius Nicolae, Slatina	629	52. Andrei Blaj, Suceava	266	81. Alpár Perini, Sf. Gheorghe	131
24. Lucian Suditu, Vaslui	618	53. Adrian Balașko, Zalău	261	81. Ioan Stan, Aiud	131
25. Florin Antonescu, Băicoi	602	54. Alex. Tomescu, București	260	83. Vasile Răcășan, Codlea	130
26. C-tin Jucovschi, R. Moldova	595	55. Alexandru Moșoi, Bacău	256	84. Miron Emilian, Bacău	125
27. Andrei Ion, Constanța	573	56. Csaba András, Oradea	246	85. Vlad Bălan, Bacău	120
28. Răzvan Albu, București	562	57. Angelo Lupuț, Satu Mare	235	86. Mihail Damboiu, București	117
29. C. Bojinovici, București	545	58. Vlad Dascălu, Bacău	226	87. D. Scripniciuc, Timișoara	114
30. Alin Coman, Târgu Jiu	507	59. Călin Ciutu, București	214	87. Ștefan Nour, Piatra Neamț	114
31. Mircea Pasoi, Ploiești	447	60. Horea Caramizaru, Craiova	212	89. Andrei Gheorghe, București	111
32. Alexandru Sofronia, Bacău	438	61. Asif-ul Haque, Bangladej	211	90. Alexandru Ivan, București	109
33. Marius Moraru, Buftea	432	62. Răzvan Tigaeru, Călărași	210	90. Ciprian Baicu, București	109
34. Silviu Hagea, Călărași	421	63. Andrei Csibi, Cluj-Napoca	200	92. Mircea-Șerban Savin, Iași	106
35. Slotta Johannes, Germania	408	63. Andrei Păduraru, Vaslui	200	93. Cosmin Răianu, Constanța	104
36. Victor Costan, București	397	63. Sorin Balasioiu, Craiova	200	93. Ionuț Ene, Turturești	104
37. Adrian Nasui, Slobozia	395	66. Alex. Constantin, Pitești	196	93. Victor Vernescu, Constanța	104
38. Constantin Răuțu, Botoșani	376	67. Alexandru Stan, Ploiești	193	96. Laura Drăgoi, Prahova	101
39. Marian Raica, Drăgutești	374	68. Eugen Sorbalo, R. Moldova	177		

Algoritmi de TARE parcurgere

Radu Vișinescu

În numărul 11/1 (ianuarie 2001) al revistei a fost publicat un articol în care a fost introdusă noțiunea de tare-parcurgere a unui graf și au fost prezentați câțiva algoritmi pentru realizarea acestei operații pentru grafuri care reprezintă labirinturi. În cadrul acestui articol vom generaliza acești algoritmi pentru a-i folosi pentru toate tipurile de grafuri.

Introducere

Un labirint este format din camere și coridoare de trecere (uși). Presupunem că într-o anumită cameră se află o persoană. În acest moment persoana nu are acces decât la informațiile pe care le poate obține în camera respectivă. Printre acestea se află informații referitoare la existența ușilor către camerele vecine.

Persoana poate prelucra informațiile pe un calculator pe măsură ce se deplasează prin labirint în scopul culegerii de date. Așadar, vom privi un labirint ca fiind un graf cu informație incompletă. Nodurile grafului vor fi reprezentate de camere, în timp ce muchiile vor fi reprezentate de ușile dintre acestea.

Definiție

Se numește *labirint* un graf neorientat $G = (X, M)$, unde X este mulțimea nodurilor și M este mulțimea muchiilor, împreună cu o submulțime de noduri distincte, $X' \subseteq X$, numite noduri de plecare.

În cadrul acestui articol vom presupune că graful este conex și există un singur nod de plecare.

În cazul grafurilor o parcurgere reprezintă o afișare (sau alt tip de prelucrare) a tuturor nodurilor, fiecare nod fiind vizitat o singură dată. Pentru un labirint putem introduce în mod natural noțiunea de tare-parcurgere.

Definiție

Se numește *tare-parcurgere* un ciclu care conține toate nodurile grafului împreună cu fixarea numărului de ordine a fiecărui nod în succesiunea din cadrul ciclului.

Astfel se extinde noțiunea de *parcurgere*; cu alte cuvinte, deplasarea se va efectua "din cameră în cameră". Dacă avem dată o anumită parcurgere, o tare-parcurgere care vizitează în aceeași ordine nodurile grafului se va numi *tare-parcurgere* conformă cu respectiva parcurgere. Dacă avem pentru început doar circuitul tare-parcurgerii și labirintul are un singur nod de plecare, putem forma o primă tare-parcurgere prin vizitarea fiecărui nod în momentul în care intrăm pentru prima dată în camera corespunzătoare lui.

Tare-parcurgeri ale vârfurilor

În continuare vom referi tare-parcurgerea definită anterior ca fiind o *tare-parcurgere a vârfurilor*. Ne punem problema dacă în cazul unor parcurgeri clasice putem găsi algoritmi care generează o tare-parcurgere conformă respectivei parcurgeri.

Pargurgerile DF și DL

S-a prezentat în [1] algoritmul tare-parcurgerii conforme parcugerii *DF* în cazul particular al unui labirint pe patru direcții în plan. Prezentăm acum algoritmul pentru cazul general al unui graf:

```
procedură TARE_DF(i)
  vizitează(i)
  marcaj(i) ← 1
  pentru toți vecinii j ai lui i
  execută
    dacă marcaj(j) = 0
    atunci
      TARE_DF(j)
      vizitează(i)
    sfârșit dacă
  sfârșit pentru
sfârșit procedură
```

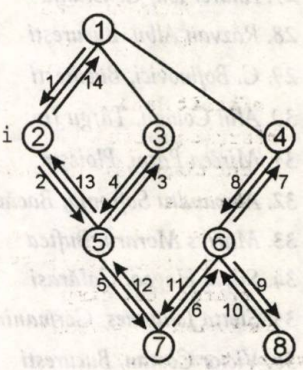


Figura 1

Pentru graful din figura 1 circuitul (ciclul) tare-parcurgerii este 1, 2, 5, 3, 5, 7, 6, 4, 6, 8, 6, 7, 5, 2, 1.

Putem însă să ne imaginăm că vizitarea propriu-zisă are loc la ultima trecere prin fiecare cameră. De exemplu, ar putea dori să mănânce labirintul cameră cu cameră și să adune gunoiul în camera de plecare. Obținem astfel un alt tip de parcurgere a grafului pe care o vom numi *DL* (*Deapth Last*).

```
procedură DL(i)
  marcaj(i) ← 1;
  pentru toți vecinii j ai lui i execută
  dacă marcaj(j) = 0 atunci
      DL(j)
  sfârșit dacă
 sfârșit pentru
  vizitează(i)
sfârșit procedură
```

Parcurgerea *DL* corespunzătoare grafului din figura 1 este 3, 4, 8, 6, 7, 5, 2, 1. În cazul în care graful este arbore, parcurgerea *DL* reprezintă o traversare în postordine.

Parcurgerile D și DD

Prezentăm în continuare o variantă a algoritmului care realizează parcurgeri *D* (*Depth*) și *DD* (*Double Depth*). Vom folosi două stive de lucru: stiva primară (*SP*) și stiva finală (*SF*). Nodurile grafului vor fi inserate succesiv în cele două stive, iar algoritmul se încheie în momentul în care stiva finală devine vidă.

```
procedură TARE_D(i)
  SF ← vidă
  SP ← vidă
  inserează i în SP
  marcaj(i) ← 1
  repetă
    inserează i în SP
    vizitează(i)          // (1)
    inserează i în SF
    pentru toți vecinii j ai lui i execută
      dacă marcaj(j) = 0 atunci
        vizitează(j)      // (2)
        inserează j în SP
        marcaj(j) ← 1
        vizitează(i)
    sfârșit dacă
  sfârșit pentru
    cât timp (vârf(SP) nu este adiacent cu vârf(SF))
    execută
      inserează i în SF
      vizitează(vârf(SF))
    sfârșit cât timp
  până când (SF = vidă)
sfârșit procedură
```

În figura 2 este prezentată o porțiune din circuitul tare-parcurgerii împreună cu modificările realizate până în acel moment în cele două stive:

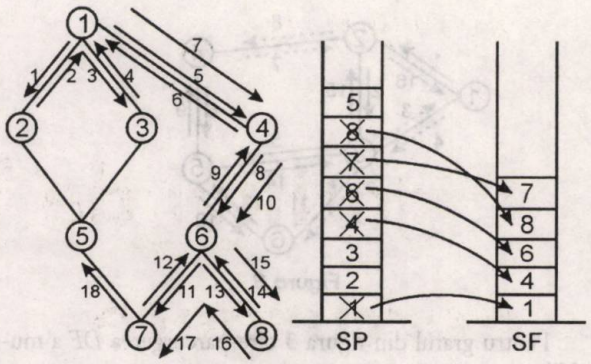


Figura 2

Dacă vizitarea nodurilor are loc doar în momentul inserării lor în prima stivă (linia (1)), atunci avem o parcurgere *D*. Dacă nodurile sunt vizitate numai în momentul inserării lor în stiva finală (linia (1)), avem un nou tip de parcurgere pe care am denumit-o *Double Depth*.

Pentru acest graf, o parcurgere *D* este 1, 2, 3, 4, 6, 7, 8, 5. Parcurgerea *DD* a aceluiași graf este 1, 4, 6, 8, 7, 5, 3, 2.

Încheiem această secțiune cu următoarea remarcă: de cele mai multe ori executarea unei funcții recursive este echivalentă cu o tare-parcurgere a vârfurilor arborelui asociat, tare-parcurgere conformă cu parcurgerile în preordine, inordine sau postordine ([2]).

Tare-parcurgeri ale muchiilor

Vom introduce acum noțiunea de tare-parcurgere a muchiilor unui graf.

Definiție

Se numește *tare-parcurgere a muchiilor* un ciclu care trece cel puțin o dată prin toate muchiile grafului.

Nu ne mai punem imediat problema generalizării unei parcurgeri; din definiție lipsește fixarea numărului de ordine pentru nodurile sau muchiilor în circuit (ciclu).

Trebuie observat faptul că această noțiune reprezintă o liniarizare a unui graf. Cu alte cuvinte reprezintă o stocare sub formă liniară a tuturor informațiilor necesare pentru refacerea grafului inițial. Prezentăm un prim algoritm pentru tare-parcuregerea muchiilor, obținut prin modificarea algoritmului de parcurgere în adâncime:

```
procedură TARE_MUCHII(i)
  vizitează(i)
  marcaj(i) ← 1
  pentru toți vecinii j ai lui i execută
  dacă marcaj(j) = 0 atunci
      TARE_MUCHII(j)
      vizitează(i)
    altfel
      vizitează(j)
      vizitează(i)
  sfârșit dacă
 sfârșit pentru
sfârșit procedură
```

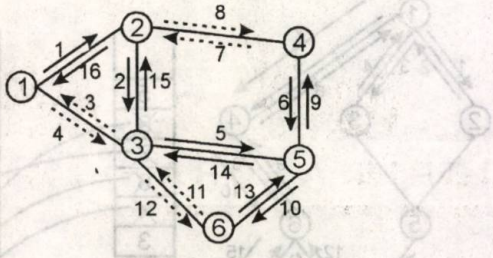


Figura 3

Pentru graful din figura 3 tare-parcurgerea *DF* a muchiilor este următoarea: 1, 2, 3, 1, 3, 5, 4, 2, 4, 5, 6, 3, 6, 5, 3, 2, 1.

Determinarea unei baze de cicluri fundamentale

Prin *baza de cicluri fundamentale* a unui graf înțelegem o mulțime maximală de cicluri în care fiecare ciclu conține cel puțin o muchie care nu face parte din nici unul dintre celelalte cicluri.

Pentru determinarea bazei de cicluri fundamentale a unui graf vom începe cu o tare-parcurgere *DF* a muchiilor grafului.

În continuare vom prelucra vectorul de noduri obținut astfel:

```
procedură cicluri(x:vector; l:lungime)
  variabile aux:vector
            k:întreg
  k ← 0
  pentru i ← 1, l execută
    k ← k + 1
    aux(k) ← v(i)
    găsit ← fals
    pentru j ← 1, k - 1 execută
      dacă aux(j) = aux(i) atunci
        găsit ← adevărat
        poz ← j
    sfârșit dacă
  sfârșit pentru
  dacă găsit atunci
      dacă poz = k - 2 atunci
          k ← k - 2 // un ciclu trebuie să conțină cel
                       puțin două muchii
        altfel
        scrie 'Ciclul:'
          pentru x ← poz + 1, k execută
            scrie aux(x), ' '
          sfârșit pentru
    sfârșit dacă
  sfârșit dacă
 sfârșit pentru // k va avea valoarea 1
sfârșit procedură
```

În figura 4 este prezentat conținutul vectorului aux în diverse puncte ale execuției algoritmului pentru graful din figura 3.

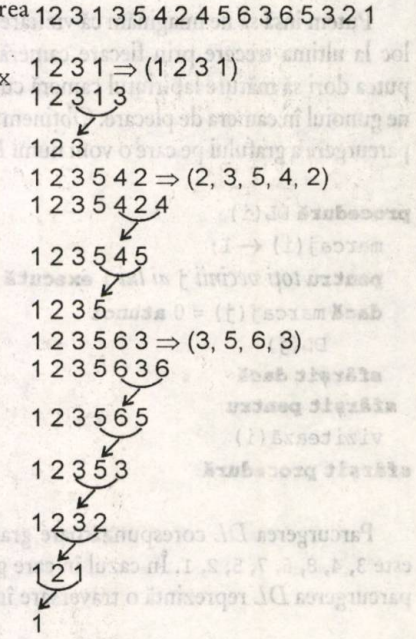


Figura 4

Problema comis-voiajorului chinez

În [3] este analizată *problema comis-voiajorului chinez*. Aceasta poate fi enunțată astfel: *Fie un graf neorientat cu costuri pe muchii; să se determine un traseu care trece cel puțin o dată prin fiecare muchie și are costul minim.*

Problema se reduce la determinarea unei tare-parcurgeri de cost minim a muchiilor. În [3] este prezentată o rezolvare în timp polinomial a acestei probleme.

Este firesc să ne punem întrebarea dacă există o astfel de rezolvare polinomială în cazul în care impunem condiția ca circuitul să treacă cel puțin o dată prin fiecare nod. Cu alte cuvinte, dorim să găsim un algoritm polinomial cu ajutorul căruia se poate determina o tare-parcurgere optimă a vârfurilor.

Dacă am găsi un algoritm polinomial de rezolvare a acestei probleme, am avea, practic, un algoritm polinomial pentru determinarea unui ciclu hamiltonian de cost minim. Așadar, problema se reduce la una *NP-completă*.

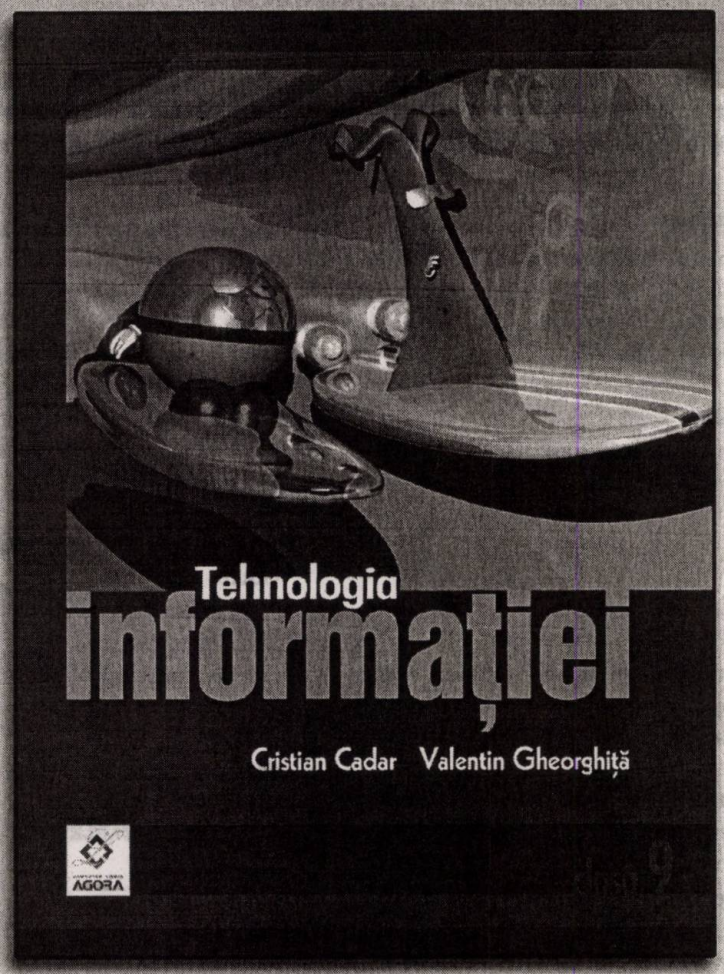
Analiza complexității

Pentru a studia complexitatea algoritmilor descriși alegem ca operație fundamentală deplasarea unei persoane dintr-o celulă în alta. În cazul tare-parcurgerii *DF* realizăm $2 \cdot (n - 1)$ astfel de deplasări, deci ordinul de complexitate este $O(n)$, unde n este numărul nodurilor. Pentru tare-parcurgerea *D* sunt necesare $4 \cdot (n - 1)$ deplasări, deci ordinul de complexitate este tot $O(n)$. Pentru tare-parcurgerea *DF* a muchiilor avem nevoie de $2 \cdot m$ deplasări, deci ordinul de complexitate este $O(m)$, unde m este numărul muchiilor.

Bibliografie

1. R. Vișinescu, *Algoritmi de tare-parcurgere*, Gazeta de informatică, ianuarie 2001
2. L. Livovschi, H. Georgescu, *Algoritmi. Elaborare și complexitate*
3. N. Christofides, *Graph theory: An algorithmic approach*

COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel.0264 - 192422

COMPUTER LIBRIS
AGORA

Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

În cadrul acestui articol vom prezenta operatorii pe care îi avem la dispoziție în PHP, precum și cele mai importante structuri de control pe care le putem folosi.

Operatori în PHP

Interpretorul *PHP* permite folosirea a nouă tipuri diferite de operatori. Aceștia operează asupra unor expresii (una, două sau trei) și furnizează ca rezultat o altă expresie (rezultatul operației corespunzătoare).

Operatori aritmetici

Toți operatorii aritmetici operează asupra a două expresii. (operanzi). Există cinci astfel de operatori:

- adunare (`'+'`);
- scădere (`'-'`);
- înmulțire (`'*'`);
- împărțire (`'/'`);
- rest (`'%'`).

Aceste operații se aplică asupra unor valori care reprezintă tipuri numerice. Dacă unul dintre operanzi nu are tip numeric, atunci el va fi convertit automat la o valoare întreagă.

Dacă cel puțin unul dintre operanzi (după efectuarea eventualelor conversii necesare) este un număr real, atunci rezultatul operației va fi tot un număr real, cu excepția operatorului rest; rezultatul operației de determinare a restului este întotdeauna un număr întreg. Dacă ambii operanzi sunt numere întregi, atunci rezultatul va fi un număr întreg, cu excepția împărțirii al cărei rezultat este întotdeauna un număr real.

Dacă operatorul rest este aplicat asupra unor numere reale, atunci rezultatul este un număr întreg.

Formula de calcul a restului împărțirii celor două numere este $a \% b = \mathbf{sgn}(a) \cdot \mathbf{sgn}(b) \cdot [\{|a| / |b|\} \cdot |b|]$, unde:

- $|x|$ reprezintă modulul valorii x; funcția modul aplicată asupra unei valori x furnizează ca rezultat valoarea x, dacă x este un număr pozitiv și valoarea $-x$, dacă x este un număr negativ;
- $\mathbf{sgn}(x)$ reprezintă semnul valorii x; funcția semn aplicată asupra unei valori x furnizează ca rezultat valoarea 1 dacă x este un număr pozitiv, valoarea -1 dacă x este un număr negativ și valoarea 0 dacă x are valoarea 0;
- $[x]$ reprezintă partea întreagă a valorii x, adică cel mai mare număr întreg care este mai mic sau egal cu x;
- $\{x\}$ reprezintă partea fracționară a valorii x, adică diferența dintre x și partea sa întreagă.

Nu este permisă împărțirea la valoarea 0. Așadar, dacă al doilea operand asupra căruia se aplică unul dintre operatorii `'/'` sau `'%'` are valoarea 0, va fi semnalată o eroare.

Prezentăm în continuare un document *PHP* care exemplifică utilizarea acestor operatori asupra unor numere întregi și asupra unor numere reale. Pagina *Web* generată este prezentată în figura 1.

```
<HTML>
  <HEAD>
    <TITLE>
      Operatori
      aritmetici
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE><TR><TD width = 150>";
      $a = 7;
      $b = 4;
      echo "<TT>$a + $b = ";
      echo $a + $b;
      echo "<BR>$a - $b = ";
      echo $a - $b;
      echo "<BR>$a * $b = ";
      echo $a * $b;
      echo "<BR>$a / $b = ";
      echo $a / $b;
      echo "<BR>$a % $b = ";
      echo $a % $b;
      echo "</TT></TD><TD>";
      $a = 24.57;
      $b = 5;
      echo "<TT>$a + $b = ";
      echo $a + $b;
      echo "<BR>$a - $b = ";
      echo $a - $b;
      echo "<BR>$a * $b = ";
```

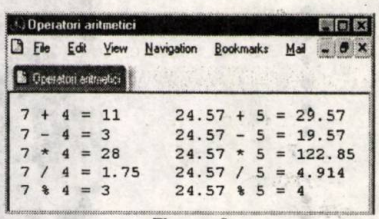


Figura 1

```
    echo $a * $b;
    echo "<BR>$a / $b = ";
    echo $a / $b;
    echo "<BR>$a % $b = ";
    echo $a % $b;
    echo "</TT></TD></TR></TABLE>";
  ?>
 </BODY>
</HTML>
```

Operatori pentru şiruri de caractere

Singurul operator care poate fi aplicat asupra şirurilor de caractere este cel de concatenare (`'.'`). Rezultatul concatenării a două şiruri de caractere este un şir de caractere care conţine primul şir, urmat de al doilea.

Un exemplu de utilizare a acestui operator este prezentat în continuare (pagina *Web* generată este cea din figura 2):

```
<HTML>
 <HEAD>
  <TITLE>
   Operatorul de concatenare
  </TITLE>
 </HEAD>
 <BODY>
  <?PHP
   $a = "Hello ";
   $b = "World!";
   echo $a.$b;
  ?>
 </BODY>
</HTML>
```

Figura 2

Operatori pe biţi

Interpretorul *PHP* pune la dispoziţie şase operatori care operează asupra biţilor unui număr întreg sau ai unui şir de caractare. Aceştia sunt:

- conjuncţie (`'&'`) - *ŞI* (*AND*);
- disjuncţie (`'|'`) - *SAU* (*OR*);
- disjuncţie exclusivă (`'^'`) - *SAU* exclusiv (*XOR*);
- negaţie (`'~'`) - *NU* (*NOT*);
- deplasare la stânga (`'<<'`);
- deplasare la dreapta (`'>>'`).

Operatorul `'&'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă ambii biţi corespunzători ai celor doi operanzi sunt setaţi pe valoarea 1. Toţi ceilalţi biţi vor avea valoarea 0.

Operatorul `'|'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă cel puţin unul dintre biţii corespunzători ai celor doi operanzi este setat pe valoarea 1. Toţi ceilalţi biţi vor avea valoarea 0.

Operatorul `'^'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă exact unul dintre biţii corespunzători ai celor doi operanzi este setat pe valoarea 1. Toţi ceilalţi biţi vor avea valoarea 0.

Operatorul `'~'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă bitul corespunzător operandului este setat pe valoarea 0. Toţi ceilalţi biţi vor avea valoarea 0.

Operatorul `'<<'` are ca rezultat o valoare întreagă în care biţii primului operand sunt deplasaţi la stânga cu numărul de poziţii indicat de al doilea operand. Poziţiile de la sfârşitul reprezentării sunt completate cu zerouri.

Operatorul `'>>'` are ca rezultat o valoare întreagă în care biţii primului operand sunt deplasaţi la dreapta cu numărul de poziţii indicat de al doilea operand. Poziţiile de la începutul reprezentării sunt completate cu valoarea bitului semn.

Pentru operatorii de deplasare, dacă al doilea operand are valoarea mai mare decât numărul de biţi folosiţi pentru reprezentarea numerelor întregi, atunci numărul poziţiilor cu care sunt deplasaţi biţii primului operand va fi dat de restul împărţirii valorii celui de-al doilea operand la numărul biţilor folosiţi pentru reprezentare. Dacă valoarea celui de-al doilea operand este negativă, atunci numărul de poziţii va fi dat de restul împărţirii valorii celui de-al doilea operand la numărul biţilor folosiţi pentru reprezentare la care se adună numărul biţilor folosiţi pentru reprezentare.

Pentru a clarifica modul în care se calculează numărul poziţiilor, vom prezenta câteva exemple. Menţionăm că în acest moment în versiunea *PHP* 4.3.0 pentru *Windows* (cea folosită pentru testarea documentelor *PHP* prezentate în cadrul acestui articol) sunt folosiţi 32 de biţi pentru reprezentarea numerelor întregi, dintre care unul este bitul semn.

- expresia `$a << 36` este echivalentă cu expresia `$a << 4`, deoarece `36 % 32 = 4`;
- expresia `$a << -52` este echivalentă cu expresia `$a << 12`, deoarece `-52 % 32 + 32 = -20 + 32 = 12`.

Dacă nu au loc depăşiri de reprezentare, o deplasare la stânga cu p poziţii este echivalentă cu o înmulţire cu valoarea 2^p, iar o deplasare la dreapta cu p poziţii este echivalentă cu o împărţire întreagă la valoarea 2^p.

Dacă un operand asupra căruia acţionează un operator pe biţi este număr real, acesta este convertit la o valoare întreagă.

În cele ce urmează prezentăm un document *PHP* în care sunt utilizaţi cei şase operatori pe biţi prezentaţi. Pagina *Web* generată este ilustrată în figura 3.

```
<HTML>
 <HEAD>
  <TITLE>
   Operatori pe biti
  </TITLE>
 </HEAD>
 <BODY>
  <?PHP
   $a = 14;
   $b = 3;
```

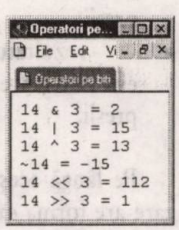


Figura 3

```
        echo "<TT>$a & $b = ";
        echo $a & $b;
        echo "<BR>$a | $b = ";
        echo $a | $b;
        echo "<BR>$a ^ $b = ";
        echo $a ^ $b;
        echo "<BR>~$a = ";
        echo ~$a;
        echo "<BR>$a << $b = ";
        echo $a << $b;
        echo "<BR>$a >> $b = ";
        echo $a >> $b;
        echo "</TT>";
    ?>
  </BODY>
</HTML>
```

Dacă operanzii asupra cărora se aplică operatori logici sunt șiruri de caractere, atunci operațiile sunt efectuate asupra codurilor *ASCII* ale fiecărui caracter în parte. Dacă în documentul *PHP* anterior înlocuim instrucțiunile

```
$a = 14;
$b = 3;
```

cu instrucțiunile

```
$a = "Hello";
$b = "World";
```

atunci pagina *Web* generată va fi cea din figura 4.

Figura 4

Operatori de comparare

Interpretorul *PHP* pune la dispoziție mai mulți operatori care pot fi folosiți pentru compararea a două valori. Expresiile în care apar astfel de operatori au ca rezultat valori logice (**true** sau **false**).

- `'=='` - rezultatul este **true** dacă cele două expresii au aceeași valoare;
- `'==='` - rezultatul este **true** dacă cele două expresii au aceeași valoare și au același tip;
- `'!='` sau `'<>'` - rezultatul este **true** dacă cele două expresii au valori diferite;
- `'!=='` - rezultatul este **true** dacă cele două expresii au valori diferite sau au tipuri diferite;
- `'<'` - rezultatul este **true** dacă valoarea primei expresii este mai mică decât valoarea celei de-a doua expresii;
- `'>'` - rezultatul este **true** dacă valoarea primei expresii este mai mare decât valoarea celei de-a doua expresii;
- `'<='` - rezultatul este **true** dacă valoarea primei expresii este mai mică sau egală cu valoarea celei de-a doua expresii;
- `'>='` - rezultatul este **true** dacă valoarea primei expresii este mai mare sau egală cu valoarea celei de-a doua expresii.

Pe lângă aceștia există și operatorul condițional `"?"` care are forma *expresie1*`?`*expresie2*`:`*expresie3* și are ca rezultat valoarea expresiei *expresie2* dacă valoarea expresiei *expresie1* este **true** (eventual după conversia la tipul `boolean`) sau valoarea expresiei *expresie3* în caz contrar.

Operatori de atribuire

Cel mai des utilizat operator de atribuire este `'='`. Acesta este folosit pentru a atribui unei variabile valoarea unei expresii. În urma operației de atribuire, tipul variabilei va deveni același cu tipul expresiei. Este obligatoriu ca primul operand să fie o variabilă. Acest operator a fost utilizat în toate exemplele anterioare.

Pe lângă acest operator, avem la dispoziție mai mulți alți operatori de atribuire. Aceștia sunt de fapt o combinație dintre operatorul `'='` și un operator aritmetic, pe biți sau pentru șiruri de caractere.

Un astfel de operator de atribuire este format din caracterul sau caracterele corespunzătoare operatorului aritmetic, pe biți sau pentru șiruri, după care urmează operatorul egal `'='`. Singurul operator pe biți care nu poate apărea într-o astfel de construcție este negația logică (`'~'`).

Primul operand asupra căruia este aplicat un astfel de operator de atribuire trebuie să fie o variabilă, iar al doilea poate fi o expresie.

În cazul folosirii unui astfel de operator de atribuire se execută operația corespunzătoare operatorului obținut prin eliminarea caracterului `'='` între valoarea variabilei care reprezintă primul operand și valoarea expresiei care reprezintă cel de-al doilea operand, iar rezultatul expresiei este atribuit variabilei care reprezintă primul operand.

De exemplu, expresia `$a += 2` are ca efect creșterea cu 2 a valorii variabilei `$a`, operația fiind echivalentă cu `$a = $a + 2`.

Rezultatul unei expresii de atribuire întotdeauna este valoarea atribuită variabilei care reprezintă primul operand.

Prezentăm în continuare toți operatorii de atribuire de acest tip, precum și expresiile echivalente:

- adunare (`'+='`): `$a += 2` ⇔ `$a = $a + 2`;
- scădere (`'-='`): `$a -= 2` ⇔ `$a = $a - 2`;
- înmulțire (`'*='`): `$a *= 2` ⇔ `$a = $a * 2`;
- împărțire (`'/='`): `$a /= 2` ⇔ `$a = $a / 2`;
- rest (`'%='`): `$a %= 2` ⇔ `$a = $a % 2`;
- conjuncție (`'&='`): `$a &= 2` ⇔ `$a = $a & 2`;
- disjuncție (`'|='`): `$a |= 2` ⇔ `$a = $a | 2`;
- disjuncție exclusivă (`'^='`): `$a ^= 2` ⇔ `$a = $a ^ 2`;
- deplasare la stânga (`'<<='`): `$a <<= 2` ⇔ `$a = $a << 2`;
- deplasare la dreapta (`'>>='`): `$a >>= 2` ⇔ `$a = $a >> 2`;
- concatenare (`'.='`): `$a .= "2"` ⇔ `$a = $a . "2"`.

Documentul *PHP* următor ilustrează modul de utilizare al tuturor operatorilor de atribuire prezentați. Pagina *Web* generată este cea din figura 5.

```
<HTML>
  <HEAD>
    <TITLE>
      Operatori de atribuire
    </TITLE>
  </HEAD>
```

```
<BODY>
  <?PHP
    $b = 24;
    $a = 57;
    echo "<TT>$a += $b = ";
    echo $a += $b;
    echo "<BR>$a -= $b = ";
    echo $a -= $b;
    echo "<BR>$a *= $b = ";
    echo $a *= $b;
    echo "<BR>$a /= $b = ";
    echo $a /= $b;
    echo "<BR>$a %= $b = ";
    echo $a %= $b;
    echo "<BR>$a &= $b = ";
    echo $a &= $b;
    echo "<BR>$a |= $b = ";
    echo $a |= $b;
    echo "<BR>$a ^= $b = ";
    echo $a ^= $b;
    echo "<BR>$a <<= $b = ";
    echo $a <<= $b;
    echo "<BR>$a >>= $b = ";
    echo $a >>= $b;
    echo "<BR>$a .= $b = ";
    echo $a .= $b;
    echo "</TT>";
  ?>
 </BODY>
</HTML>
```

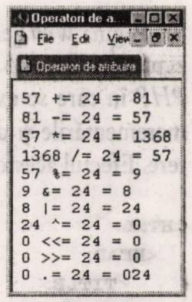


Figura 5

Operatori de incrementare și decrementare

Operatorul de incrementare (`'++'`) are ca efect creșterea cu 1 a valorii unei variable, iar cel de decrementare (`'--'`) are ca efect scăderea cu 1 a valorii variabilei. Tipul variabilei poate fi întreg sau real.

Dacă operatorul precede variabila, atunci rezultatul expresiei este valoarea obținută după incrementare sau decrementare. Dacă variabila precede operatorul, atunci rezultaul expresiei este valoarea variabilei înaintea incrementării sau decrementării.

Cu alte cuvinte, dacă operatorul precede variabila, atunci valoarea variabilei este mai întâi modificată și apoi utilizată, iar dacă variabila precede operatorul, atunci valoarea ei este mai întâi utilizată și apoi modificată.

Acești operatori pot fi utilizați și pentru variabile care conțin șiruri de caractere. Operatorul de incrementare duce la creșterea cu 1 a codului *ASCII* a ultimului caracter din șir dacă acesta este o literă sau o cifră. În cazul în care cifra este `'9'` sau litera este `'z'`, respectiv `'Z'`, atunci ea devine `'0'`, `'a'` respectiv `'A'` și se încearcă incrementarea penultimului caracter. Dacă acesta este tot `'9'`, `'z'` sau `'Z'` se aplică același procedeu și se trece la antepenultimul caracter. Procedeul continuă până în momentul în care se ajunge la un caracter care nu este `'9'`, `'z'` sau `'Z'` sau se ajunge la începutul șirului.

În acest ultim caz, la începutul șirului se adaugă caracterul `'1'`, `'a'` sau `'A'`. Incrementarea nu are nici un efect pentru caracterele care nu sunt cifre sau litere.

Decrementarea șirurilor de caractere nu are nici un efect, valorile variabilelor rămânând nemodificate. Există o singură excepție și anume șirurile care reprezintă numere întregi sau reale. Acestea sunt incrementate și decrementate potrivit regulilor pentru valori numerice.

Următorul document *PHP* ilustrează modul de utilizare al acestor operatori, precum și efectul acestora în funcție de poziția lor.

```
<HTML>
 <HEAD>
   <TITLE>
     Incrementare si decrementare
   </TITLE>
 </HEAD>
 <BODY>
  <?php
    $a = 7;
    echo "<TT>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>a++</TT> <I>este</I> <TT>";
    echo $a++;
    echo "</TT>. <I>Noua valoare a variabilei
       </I> <TT>a</TT> <I>este</I> <TT>$a.";
    echo "<BR>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>++a</TT> <I>este</I> <TT>";
    echo ++$a;
    echo "</TT>. <I>Noua valoare a variabilei
       </I> <TT>a</TT> <I>este</I> <TT>$a.";
    echo "<BR>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>a--</TT> <I>este</I> <TT>";
    echo $a--;
    echo "</TT>. <I>Noua valoare a variabilei
       </I> <TT>a</TT> <I>este</I> <TT>$a.";
    echo "<BR>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>--a</TT> <I>este</I> <TT>";
    echo --$a;
    echo "</TT>. <I>Noua valoare a variabilei
       </I> <TT>a</TT> <I>este</I> <TT>$a.";
    $a = 7.5;
    echo "<BR>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>a++</TT> <I>este</I> <TT>";
    echo $a++;
    echo "</TT>. <I>Noua valoare a variabilei
       </I> <TT>a</TT> <I>este</I> <TT>$a.";
    echo "<BR>a = $a";
    echo ": </TT><I>Rezultatul operatiei
         </I> <TT>++a</TT> <I>este</I> <TT>";
    echo ++$a;
```

```
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>a--</TT> <I>este</I> <TT>";
        echo $a--;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>--a</TT> <I>este</I> <TT>";
        echo --$a;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        $a = "PHP";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>a++</TT> <I>este</I> <TT>";
        echo $a++;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>++a</TT> <I>este</I> <TT>";
        echo ++$a;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>a--</TT> <I>este</I> <TT>";
        echo $a--;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
              </I> <TT>--a</TT> <I>este</I> <TT>";
        echo --$a;
        echo "</TT>. <I>Noua valoare a variabilei
              </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "</TT>";
    ?>
  </BODY>
</HTML>
```

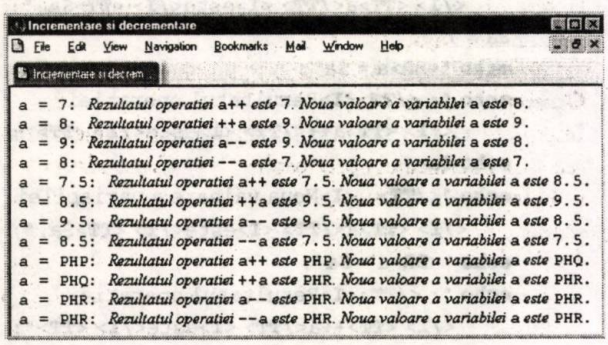


Figura 6

Pagina *Web* generată de acest document *PHP* este prezentată în figura 6. Prezentăm în continuare un document *PHP* în care se exemplifică modul în care sunt realizate incrementările și decrementările pentru șirurile de caractere. Efectul acestor operații este prezentat în figura 7.

```
<HTML>
  <HEAD>
    <TITLE>
      Siruri de caractere
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE width = 100%>";
      $a = "7";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "7.5";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "7.5z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "7.9z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "7zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "az99zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "zz99zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "99Z99Z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a.".</TT>
                                  <BR></TD></TR>";
      $a = "+";
```

```
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "+9";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "a+9";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "++9";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "-9";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "-9z";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "--9";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "+-99";
        echo "<TR><TD width = 50%>Inainte de
         incrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
        $a = "-9";
        echo "<TR><TD width = 50%>Inainte de
         decrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa decrementare: <TT>a = ".--$a."</TT>
                                <BR></TD></TR>";
        $a = "-9.5";
        echo "<TR><TD width = 50%>Inainte de
         decrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa decrementare: <TT>a = ".--$a."</TT>
                                <BR></TD></TR>";
        $a = "-9.5b";
        echo "<TR><TD width = 50%>Inainte de
         decrementare: <TT>a = $a</TT>. </TD><TD>
         Dupa decrementare: <TT>a = ".--$a."</TT>
                        <BR></TD></TR></TABLE>";?>
    </BODY>
</HTML>
```

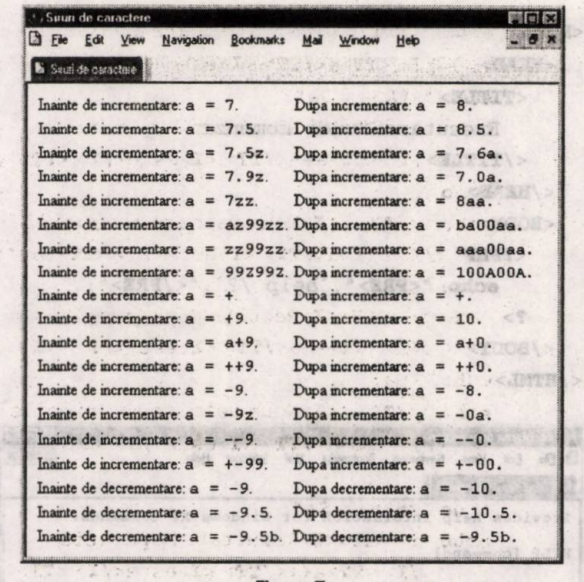


Figura 7

Operatori logici

Există patru tipuri de operatori care pot fi folosiți pentru construirea expresiilor logice. Operanzii trebuie să aibă tipul `boolean`; dacă operatorii au alt tip, valoarea este convertită la o valoare booleană. Operatorii logici sunt:

- conjuncție logică (`'&&'` sau `'and'`) - *ȘI (AND)*;
- disjuncție logică (`'||'` sau `'or'`) - *SAU (OR)*;
- disjuncție exclusivă logică (`'xor'`) - *SAU* exclusiv (*XOR*);
- negație logică (`'!'`) - *NU (NOT)*.

Expresia în care apar operatorii `'&&'` și `'and'` va avea valoarea `true` dacă ambii operanzi au valoarea **`true`**.

Expresia în care apar operatorii `'||'` și `'or'` va avea valoarea **`true`** dacă cel puțin unul dintre operanzi are valoarea **`true`**.

Expresia în care apare operatorul `'xor'` va avea valoarea **`true`** dacă exact unul dintre operanzi are valoarea **`true`**.

Expresia în care apare operatorul `'!'` va avea valoarea **`true`** dacă operandul are valoarea **`false`**.

În celelalte cazuri valorile acestor tipuri de expresii vor fi **`false`**.

Operatori pentru controlul erorilor

În *PHP* există un operator (`'@'`) care permite ignorarea erorilor. Dacă este aplicat asupra unei expresii care ar duce la afișarea unui mesaj de eroare, atunci mesajul respectiv nu va mai fi afișat.

Operatori de execuție

Interpretorul *PHP* permite executarea unor comenzi sistem prin intermediul operatorului `"``"`. Comanda respectivă este cuprinsă între apostroafele inverse, iar rezultatul acestei comenzi este un șir de caractere care reprezintă și rezultatul expresiei.

Următorul *script PHP* determină execuția pe *server* a comenzii `help /?` și afișarea rezultatului în fereastra programului de navigare. Efectul este ilustrat în figura 8.

```
<HTML>
  <HEAD>
    <TITLE>
      Executarea unei comenzi
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<PRE>".`help /?`."</PRE>";
    ?>
  </BODY>
</HTML>
```

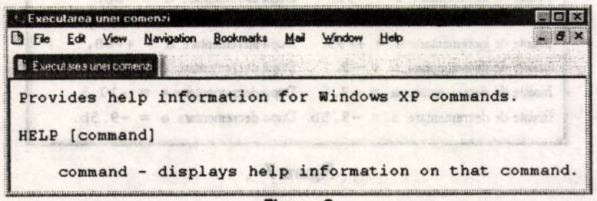


Figura 8

Operatori pentru vectori

Singurul operator care poate fi aplicat asupra vectorilor este cel de concatenare (`'+'`). Rezultatul concatenării a doi vectori este un vector care conține elementele din cei doi vectori.

Dacă cei doi vectori conțin elemente cu aceeași cheie, atunci este păstrat doar elementul din vectorul care reprezintă primul operand.

Un exemplu de utilizare a acestui operator este prezentat în continuare (pagina *Web* generată este cea din figura 9):

Figura 9

```
<HTML>
  <HEAD>
    <TITLE>
      Concatenarea vectorilor
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $a[2] = 2;
      $a[4] = 4;
      $a[5] = 5;
      $a[7] = 7;
      $b[1] = 100;
      $b[2] = 200;
      $b[3] = 300;
      $b[4] = 400;
      $b[5] = 500;
      $c = $a + $b;
      echo "$c[0] $c[1] $c[2] $c[3] $c[4] $c[5]
                                        $c[7]";
    ?>
  </BODY>
</HTML>
```

Ordinea operațiilor

Evident, o expresie poate conține mai mulți operatori, din categorii diferite. Pentru a indica ordinea în care trebuie efectuate calculele, trebuie stabilită o ordine a operațiilor. Pentru aceasta a fost definită *precedența* și *asociativitatea* operatorilor.

Valoarea expresiilor corespunzătoare operatorilor cu precedență mai mare va fi calculată înaintea valorilor expresiilor cu o precedență mai mică. De exemplu, pentru expresia `1 + 2 * 3` va fi efectuată mai întâi înmulțirea `2 * 3`, datorită faptului că operatorul `'*'` are precedența mai mare decât cea a operatorului `'+'`.

Dacă o expresia sau o subexpresie conține numai operatori cu aceeași precedență, atunci operațiile se vor efectua în ordinea determinată de asociativitatea acestor operatori care poate fi de la dreapta la stânga sau de la stânga la dreapta. Toți operatorii cu aceeași precedență au același tip de asociativitate.

Dacă asociativitatea este de la dreapta la stânga, atunci operațiile se vor efectua începând din "partea dreaptă" a expresiei, iar dacă asociativitatea este de la stânga la dreapta, vor fi efectuate mai întâi operațiile aflate în "partea stângă" a expresiei.

Există și operatori care nu sunt asociativi, deci nu pot exista expresii în care apar doi astfel de operatori.

Pe lângă operatorii amintiți există și alții care au funcții speciale. Pe unii i-am prezentat în cadrul articolului din numărul anterior al revistei (de exemplu cei folosiți pentru realizarea conversiilor de tip), iar pe ceilalți îi vom prezenta atunci când îi vom folosi pentru a realiza diferite operații în care aceștia pot apărea.

Evident, pentru a modifica ordinea implicită de efectuare a operațiilor pot fi folosite paranteze pentru a indica faptul că o anumită operație trebuie efectuată înaintea altora, chiar dacă acestea din urmă au o precedență mai mare.

Trebuie observat faptul că, datorită asociativității de la dreapta spre stânga a operatorilor de atribuire, expresiile de tipul `$a = $b = 1` sunt valide. Mai întâi se va efectua atribuirea `$b = 1`; variabila `$b` va primi valoarea `1` și, datorită faptului că rezultatul unei operații de atribuire este valoarea atribuită variabilei care reprezintă primul operand, în continuare se va efectua atribuirea `$a = 1`, deci și variabila `$a` va primi tot valoarea `1`.

Așadar, în urma executării unei expresii de tipul `$v1 = $v2 = ... = $vn` = *val*, toate cele *n* variabile vor avea valoarea *val*.

Tabelul 1 conține o listă cu majoritatea opratorilor din *PHP*. Liniile tabelului indică precedența, în sensul că un operator are aceeași precedență cu operatorii de pe linie în care apare și o precedență mai mare decât oricare operator aflat pe o linie anterioară.

În acest tabel se observă faptul că, deși operatorii `'&&'` și `'and'`, respectiv `'||'` și `'or'` realizează aceeași operație, ei au precedențe diferite, deci nu sunt echivalenți din toate punctele de vedere.

Tabelul 1

Tip asociativitate	Tip	Operanzi
de la stânga la dreapta	unar	. (selectare)
de la stânga la dreapta	binar	or
de la stânga la dreapta	binar	xor
de la stânga la dreapta	binar	and
de la dreapta la stânga	binar	print
de la stânga la dreapta	binar	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
de la stânga la dreapta	ternar	? :
de la stânga la dreapta	binar	||
de la stânga la dreapta	binar	&&
de la stânga la dreapta	binar	|
de la stânga la dreapta	binar	^
de la stânga la dreapta	binar	&
fără asociativitate	binar	== != === !==
fără asociativitate	binar	< <= > >=
de la stânga la dreapta	binar	<< >>
de la stânga la dreapta	binar	+ - . (concatenare)
de la stânga la dreapta	binar	* / %
de la dreapta la stânga	unar	! ~ ++ -- (int) (float) (string) (array) (object) @
de la dreapta la stânga	unar	[
fără asociativitate	unar	new

Precedența și asociativitatea operatorilor în PHP

În tabel este prezentat și numărul expresiilor asupra cărora acționează un operand. Operatorii unari acționează asupra unei singure expresii, cei binari asupra a două expresii și singurul operator ternar asupra a trei expresii.

Structuri de control în PHP

Așa cum am afirmat într-un articol anterior, instrucțiunile *PHP* sunt separate prin caracterul `';'`. Deseori, suntem nevoiți să grupăm mai multe instrucțiuni pentru a forma un bloc. Astfel, obținem instrucțiunile compuse care sunt formate din mai multe instrucțiuni simple, separate prin caracterul `';'`. În *PHP*, instrucțiunile compuse au următoarea sintaxă:

```
{
instrucțiune #1
instrucțiune #2
...
instrucțiune #n
}
```

Instrucțiunile care formează instrucțiunea compusă pot fi de orice tip: orice structură de control, alte instrucțiuni compuse etc. Așadar un bloc de instrucțiuni (o instrucțiune compusă), în *PHP*, este delimitat de acolade.

Structura if

Una dintre cele mai importante structuri în orice limbaj este cea alternativă. În *PHP* sintaxa acestei structuri este următoarea:

```
if (condiție)
  instrucțiune
```

Folosirea unei astfel de structuri indică faptul că instrucțiunea se va executa dacă și numai dacă valoarea expresiei *condiție* (eventual după conversia la tipul `boolean`) este **`true`**.

Construcția if - else

În foarte multe cazuri dorim să executăm o altă instrucțiune dacă valoarea expresiei *condiție* este **`false`**. În *PHP* putem utiliza o construcție de tipul **`if - else`** în acest scop. Sintaxa este:

```
if (condiție)
  instrucțiune #1
else
  instrucțiune #2
```

Construcția elseif

Uneori, suntem nevoiți să folosim structuri alternative imbricate în diferite scopuri. Folosind construcții de tipul **`if - else`** vom obține o secvență de tipul:

```
if (condiție #1)
  instrucțiune #1
else
 if (condiție #2)
   instrucțiune #2
 else
 ...
 else
  if (condiție #n)
       instrucțiune #n
 else
       instrucțiune #n+1
```

Limbajul *PHP* permite folosirea unei prescurtări și anume construcția **elseif**. Practic, aceasta înlocuiește un **else** urmat de un **if**. Folosind această structură, codul anterior poate fi scris astfel:

```
if (condiție #1)
   instrucțiune #1
elseif (condiție #2)
   instrucțiune #2
elseif
...
elseif (condiție #n)
   instrucțiune #n
else
   instrucțiune #n+1
```

Sintaxe alternative

Dacă dorim să se execute mai multe instrucțiuni când este îndeplinită o anumită condiție, atunci ar trebui să folosim instrucțiuni compuse. De exemplu, pentru o structură **if** vom scrie:

```
if (condiție) {
   instrucțiune #1
   instrucțiune #2
   ...
   instrucțiune #n
}
```

Limbajul *PHP* oferă o alternativă care permite indicarea tuturor instrucțiunilor fără a mai folosi acolade. Structura prezentată mai sus poate fi rescrisă astfel:

```
if (condiție):
   instrucțiune #1
   instrucțiune #2
   ...
   instrucțiune #n
endif
```

Practic, pentru orice structură de control *PHP*, putem înlocui acolada deschisă prin caracterul ':' și acolada închisă printr-un cuvânt cheie obținut prin adăugarea prefixului **end** la cuvântul cheie care indică structura de control.

Trebuie observat faptul că **else** și **elseif** nu sunt structuri, ele fiind simple construcții folosite în cadrul structurii **if**. Din acest motiv, nu vom avea niciodată **endelse** sau **endelseif**, ci doar **endif**-uri. Așadar, sintaxa alternativă pentru o structură **if** în care se folosesc construcții **else** și **elseif** este următoarea:

```
if (condiție #1)
   grup instrucțiuni #1
elseif (condiție #2)
   grup instrucțiuni #2
elseif
...
elseif (condiție #n)
   grup instrucțiuni #n
else
   grup instrucțiuni #n+1
endif;
```

Structura while

Majoritatea *script*-urilor *PHP* vor conține cicluri. Pentru a folosi o buclă anterior condiționată vom utiliza structura **while**. Sintaxa acestei proceduri este:

```
while (condiție)
   instrucțiune
```

Efectul acestei structuri este executarea instrucțiunii atât timp cât valoarea expresiei *condiție* (eventual după conversia la tipul `boolean`) este **true**. Trebuie observat faptul că este posibil ca instrucțiunea să nu fie executată niciodată. Și pentru această structură avem la dispoziție o sintaxă alternativă și anume:

```
while (condiție):
   grup instrucțiuni
endwhile;
```

Structura do - while

Uneori, dorim să folosim bucle posterior condiționate. În acest scop, în *PHP* avem la dispoziție structura **do - while** a cărei sintaxă este:

```
do
   instrucțiune
while (condiție);
```

Singura diferență față de structura **while** este faptul că valoarea condiției este determinată, de fiecare dată, după executarea instrucțiunii. Ca urmare, instrucțiunea va fi executată cel puțin o dată. Nu există o sintaxă alternativă pentru această structură.

Structura for

O alternativă cu o funcționalitate mai ridicată pentru utilizarea buclelor este structura repetitivă **for**. Sintaxa este foarte asemănătoare cu cea din limbajele *C/C++* și *Java* și anume:

```
for (expresie #1; expresie #2; expresie #3)
   instrucțiune
```

Prima expresie este evaluată o singură dată, înainte de începerea execuției ciclului. Instrucțiunea este executată cât timp cea de-a doua expresie are valoarea **true**. De fiecare dată, după executarea instrucțiunii este evaluată cea de-a treia expresie. Oricare dintre cele trei expresii poate lipsi;

în cazul în care o expresie lipsește, se consideră că ea are valoarea **true**. Pentru structura **for** poate fi folosită și următoarea sintaxă alternativă:

for (*expresie #1*; *expresie #2*; *expresie #3*) :
  instrucțiune
endfor;

Exemple

Vom prezenta în continuare un exemplu care afișează numerele cuprinse între 1 și 15. Numerele pare vor fi scrise cu roșu, iar cele impare care sunt divizibile cu 3 cu verde, iar celelalte cu albastru.

Vom folosi o structură repetitivă pentru a afișa numerele și o structură alternativă pentru a determina culoarea cu care vor fi afișate. Putem utiliza oricare dintre cele trei structuri repetitive; codul *PHP* pentru cele trei variante sunt prezentate în caseta de mai jos, iar pagina generată este cea din figura 10, indiferent de varianta utilizată.

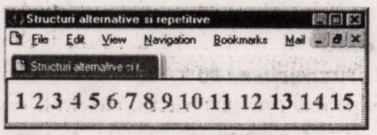


Figura 10

Structura foreach

Această structură poate fi folosită pentru a realiza o iterație printre toate elementele unui vector. Așadar, ea nu poate fi folosită decât împreună cu vectori; utilizarea sa asupra unei variabile de alt tip duce la apariția unei erori.

Există două sintaxe acceptate pentru această structură și anume:

foreach (*expresie_vectorială* **as** *$valoare*)
  instrucțiune
foreach (*expresie_vectorială* **as** *$cheie* => *$valoare*)
  instrucțiune

Dacă se utilizează prima variantă, atunci la fiecare iterație valoarea elementului curent este atribuită variabilei *$valoare* și apoi se trece la elementul următor (a cărui valoare va fi atribuită variabilei la următoarea iterație). Execuția ciclului se încheie în momentul în care nu mai există alte elemente în vector.

Singura diferență care apare în cazul utlizării celei de a doua variante este faptul că la fiecare iterație valoarea cheii elementului curent este atribuită variabilei *$cheie*.

Prezentăm în continuare un document *PHP* în care sunt utilizate cele două sintaxe ale structurii **foreach**. Efectul este ilustrat în figura 11.

```
<HTML>
  <HEAD>
    <TITLE>
      Structura foreach
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE width = 100%> <TR>
                                <TD width = 50%>";
      // un vector obisnuit
      $a = array (1, 2, 3, 10);
      echo "<B>Parcurgerea unui vector:
                                    </B><BR><OL>";
      foreach ($a as $v)
        echo "<LI><I>Valoarea curenta este
                        </I>: <TT>$v</TT>\n<BR>";
      // parcurgerea vectorului, contorizarea
      // elementelor si afisarea numarului de
      // ordine al elementului curent
      $i = 0;
      echo "</OL><B>O alta parcurgere:
                                    </B><UL>";
```

Structuri alternative și repetitive

Bucla for

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      for ($i=1;$i<=15;$i++):
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
      endfor;
    ?>
  </BODY>
</HTML>
```

Bucla while

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $i = 1;
      while ($i<=15){
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
        $i++;
      }
    ?>
  </BODY>
</HTML>
```

Bucla do - while

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $i = 1;
      do{
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
        $i++;
      } while ($i<=15);
    ?>
  </BODY>
</HTML>
```

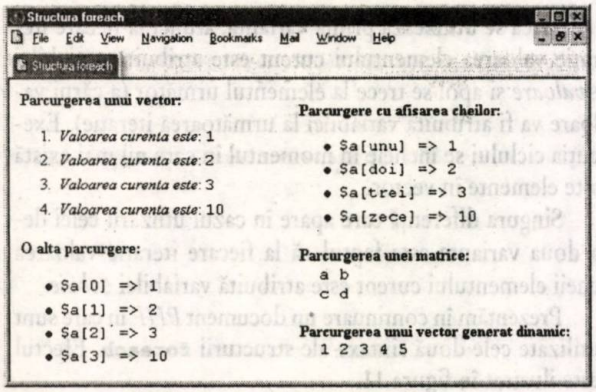


Figura 11

```
    foreach ($a as $v){
      echo "<LI><TT>\$a[$i] => $v</TT>\n
                                    <BR>";
      $i++;
    }
    // un vector ale carui elemente sunt
    // identificate prin chei
    $a = array (
      "unu" => 1,
      "doi" => 2,
      "trei" => 3,
      "zece" => 10
    );
    echo "</UL><TD width = 50%><B>
          Parcurgere cu afisarea cheilor:
                                    </B><UL>";
    foreach ($a as $k => $v) {
      echo "<LI><TT>\$a[$k] => $v</TT>\n";
    }
    // o matrice
    $b[0][0] = "a";
    $b[0][1] = "b";
    $b[1][0] = "c";
    $b[1][1] = "d";
    echo "</UL><B>Parcurgerea unei
                matrice:</B><BR><TT>";
    foreach ($b as $v1) {
      echo "  ";
      foreach ($v1 as $v2)
        echo "$v2\n";
      echo "<BR>";
    }
    echo "</TT><BR><B>Parcurgerea unui
      vector generat dinamic:</B><BR><TT>
                            ";
    // un vector generat dinamic
    foreach (array(1, 2, 3, 4, 5) as $v)
      echo "$v\n";
    echo "</TT></TD></TR></TABLE>";
  ?>
 </BODY>
</HTML>
```

Structura switch

În anumite cazuri trebuie să efectuăm anumite operații în funcție de anumite valori ale unei expresii. O soluție ar fi folosirea unor structuri **if** imbricate sau a uneia singure în care apar mai multe construcții **elseif**.

Să presupunem că o anumită variabilă poate avea cinci valori distincte și pentru fiecare valoare trebuie executată o anumită instrucțiune. Dacă folosim structura **if**, secvența de cod va fi următoarea:

```
if ($variabila == valoare #1)
  instrucțiune #1
elseif ($variabila == valoare #2)
  instrucțiune #2
elseif ($variabila == valoare #3)
  instrucțiune #3
elseif ($variabila == valoare #4)
  instrucțiune #4
else
  instrucțiune #5
```

Folosind structura **switch**, secvența echivalentă este:

```
switch ($variabila){
 case valoare #1: instrucțiune #1
                  break;
 case valoare #2: instrucțiune #2
                  break;
 case valoare #3: instrucțiune #3
                  break;
 case valoare #4: instrucțiune #4
                  break;
 case valoare #5: instrucțiune #5
                  break;
}
```

Prezentăm în continuare sintaxa generală a structurii **switch**:

```
switch (expresie){
 case valoare #1: grup instrucțiuni #1
 case valoare #2: grup instrucțiuni #2
 ...
 case valoare #n: grup instrucțiuni #n
 [default: grup instrucțiuni #n + 1]
}
```

Pentru început se determină valoarea expresiei și apoi se verifică dacă aceasta apare ca valoare pentru una dintre construcțiile **case**.

În caz afirmativ, toate instrucțiunile începând cu cea corespunzătoare valorii respective (până la sfârșitul blocului **switch**) vor fi executate.

Execuția acestor instrucțiuni poate fi întreruptă dacă este folosită instrucțiunea **break**, care va fi prezentată în cadrul acestui articol.

Dacă valoarea expresiei nu corespunde nici uneia dintre valorile corespunzătoare construcțiilor **case**, se execută instrucțiunile corespunzătoare construcției **default**. Dacă aceasta lipsește, atunci nu se execută nici o instrucțiune.

Trebuie precizat faptul că valorile corespunzătoare construcțiilor **case** pot fi numere întregi, numere reale sau șiruri de caractere; nu pot fi utilizate obiecte sau vectori.

Instrucțiunea break

Această instrucțiune poate fi folosită pentru a întrerupe forțat execuția unui ciclu sau a secvenței de instrucțiuni corespunzătoare unei structuri **switch**.

Spre deosebire de limbajele *Pascal* sau *C/C++*, instrucțiunea poate fi urmată de un argument care indică numărul de structuri imbricate a căror execuție se încheie. Valoarea implicită este 1, deci se întrerupe execuția unei singure structuri.

Următoarea secvență de cod *PHP* realizează parcurgerea elementelor unui vector de numere întregi până în momentul în care se întâlnește un număr negativ.

```
foreach ($a as $v) .
  if ($v < 0)
    break;
```

Vom exemplifica acum și cazul în care este întreruptă execuția mai multor cicluri; vom considera că parcurgem elementele unei matrice pătratice cu n elemente și n coloane până în momentul în care întâlnim o valoare nulă.

```
for ($i = 0; $i < $n; $i++)
  for ($j = 0; $j < $n; $j++)
    if (!$a[$i][$j])
      break 2;
```

Instrucțiunea **break** poate fi utilizată pentru întreruperea execuției secvențelor de instrucțiuni corespunzătoare structurilor **for**, **foreach**, **while**, **do - while** și **switch**.

Instrucțiunea continue

Această instrucțiune este folosită pentru a întrerupe execuția secvenței de instrucțiuni din interiorul unui ciclu și trecerea la următoarea iterație. În cazul instrucțiunii **for**, înainte de următoarea iterație se evaluează (execută) expresia de incrementare (*expresia* #3 din sintaxa generală).

La fel ca și în cazul instrucțiunii **break**, poate apărea un argument care indică numărul structurilor imbricate asupra cărora are efect.

Următoarea secvență *PHP* realizează afișarea elementelor unui șir de numere întregi care sunt mai mari decât 1000.

```
foreach ($a as $v){
  if ($v <= 1000)
    continue;
  echo $v;
}
```

Următorul exemplu ilustrează efectul folosirii argumentelor pentru instrucțiunea **continue**.

```
$i = 0;
while ($i++ < 5){
  echo "Ciclul #1<br>\n";
  while (1) {
    echo "  Ciclul #2<BR>\n";
    while (1) {
      echo "  Ciclul #3<BR>\n";
      continue 3;
    }
    echo "Acest mesaj nu va fi afisat
                                niciodata.<BR>\n";
  }
  echo "Nici acest mesaj nu va fi afisat
                                niciodata.<BR>\n";
}
```

Alte structuri PHP

Există mai multe alte structuri *PHP* care pot fi utilizate în anumite scopuri. Vom aminti acum câteva dintre ele și le vom prezenta detaliat în momentul în care va fi necesară utilizarea lor în diferite scopuri.

Structurile **include**, **require**, **include_once** și **require_once** pot fi utilizate pentru a "insera" anumite instrucțiuni care sunt păstrate într-un alt fișier (document). Interpretorul *PHP* consideră că secvența din fișierul inserat se află în fișierul din care s-a "comandat" inserarea în poziția în care apare structura de inserare.

O altă structură este **declare** care permite crearea unor directive de execuție.

Funcțiile *PHP* trebuie să utilizeze instrucțiunea **return** pentru a furniza un rezultat. Detalii despre utilizarea acestei instrucțiuni vor fi prezentate în cadrul următorului articol dedicat limbajului *PHP*.

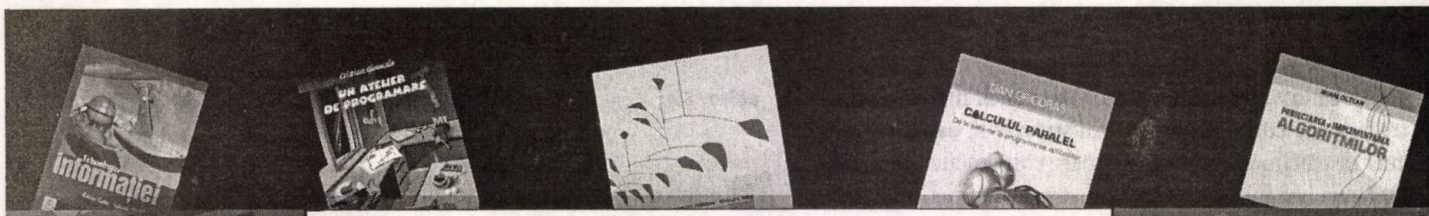
În numărul următor

Vom continua prezentarea interpretorului *PHP* cu descrierea modului în care pot fi declarate și utilizate funcții.

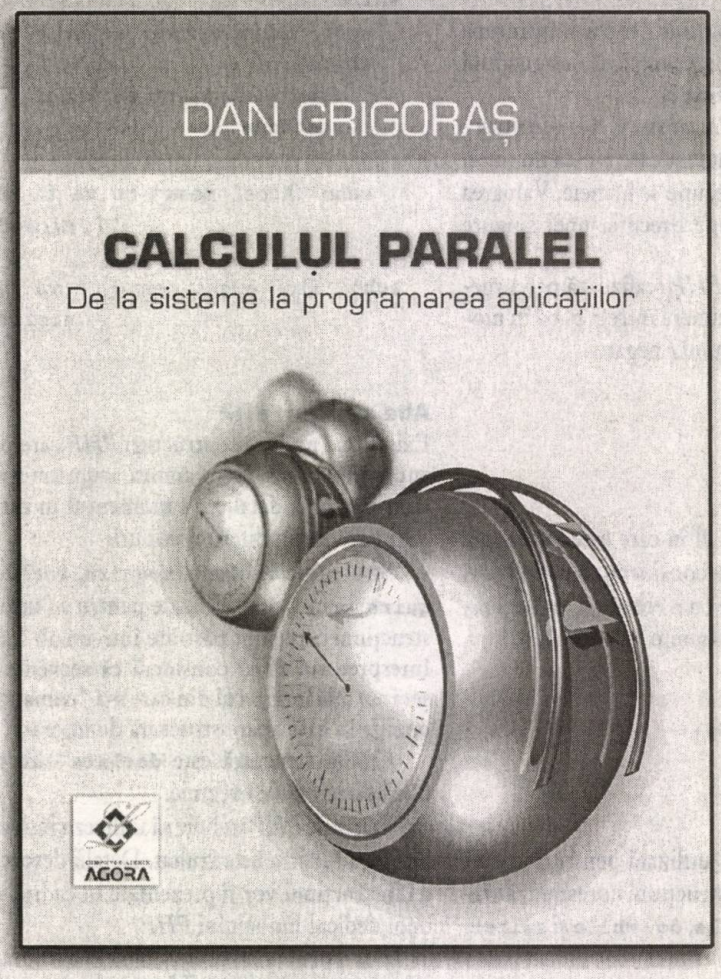
De asemenea, în următorul articol vom prezenta detalii referitoare la clasele și obiectele *PHP*. Nu vom insista asupra paradigmelor programării orientate obiect, ci vom prezenta doar modul în care pot fi utilizate clasele și obiectele.

În plus, articol din numărul următor va conține detalii despre modul în care pot fi utilizate referințele în *PHP* pentru a accesa conținutul unei singure variabile folosind mai multe nume (identificatori).

Mihai Scorțaru este redactor-șef GInfo și poate fi contactat prin e-mail la adresa **skortzy@yahoo.com**. *Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa* **csoroiu@yahoo.com**.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate, dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS
AGORA

Conceptul de ALGORITM - abordare MODERNĂ

Marin Vlada

Dezvoltarea informaticii actuale se datorează cercetărilor, rezultatelor și experiențelor din domeniile sistemelor de calcul, algoritmicii și programării, dar mai ales a interdependenței acestor domenii prin așa-numita triadă "sistem de calcul - algoritmică - programare". La baza acestei interdependențe se află conceptul de algoritm, concept care a construit pentru om o nouă filosofie: "gândirea algoritmică". Această gândire algoritmică a făcut posibilă apariția și dezvoltarea Tehnologiei Informației (IT) care reprezintă de fapt implementarea filosofiei procesării, gestionării și transmiterii informațiilor.

În cele ce urmează vom realiza o abordare modernă a conceptului de algoritm și vom evidenția necesitatea eliminării abordărilor vechi care, din păcate, mai există în multe manuale adresate elevilor și studenților.

Mulți dintre autorii manualelor de informatică din țara noastră greșesc și dau o importanță minoră conceptului de algoritm, atunci când consideră că "*algoritmul este o noțiune primară*".

Dezvoltarea unei gândiri algoritmice se poate realiza doar prin activități de instruire și formare care să aibă la bază conceptul fundamental de ***algoritm*** într-o abordare modernă și nu una învechită. Abordarea propusă va putea fi adaptată la orice nivel de instruire și la orice arie curriculară.

Introducere

Întâmplător sau nu, deceniul 7 al secolului al XX-lea a fost unul al marilor schimbări în domeniul informaticii și al sistemelor de calcul:

- ***inventarea microprocesorului*** ("*bijuteria de bază*" a sistemelor de calcul actuale) ca urmare a rezultatelor din trei domenii fundamentale: sisteme, circuite integrate și microprogramare; a urmat construirea și răspândirea pe scara largă a sistemelor de calcul de tip *PC* (*Personal Computer*), impulsionarea dezvoltării rețelelor de calculatoare, precum și apariția și dezvoltarea de noi sisteme de operare; performanțe sporite ale dispozitivelor de *intrare / ieșire*;
- ***supremația și răspândirea structurilor de control*** în *algoritmică* și *programare*; apariția *limbajului pseudocod* pentru reprezentarea și elaborarea algoritmilor; conceperea și scrierea primelor limbaje de programare care implementează *structurile de control*, adaptarea continuă a limbajelor de programare prin implementarea *structurilor de control*, a *structurilor de date*, precum și a *facilităților programării orientate obiect* (*OOP - Object Oriented Programming*);
- ***succese spectaculoase în domeniul inteligenței artificiale*** prin construirea primelor sisteme expert; conceperea și scrierea primului limbaj de programare logică (limbajul *Prolog*) care oferă suportul ***programării declarative***; dezvoltarea și utilizarea largă a metodelor și tehnicilor *inteligenței artificiale* în rezolvarea câtorva dintre cele mai dificile probleme;
- ***delimitarea problemelor*** rezolvate cu calculatorul (*probleme decidabile*) în două clase distincte: clasa problemelor rezolvate prin *metode imperative* (*procedurale*) și clasa problemelor rezolvate prin *metode declarative*; delimitarea clasei *problemelor nedecidabile*; apariția și dezvoltarea tehnicilor pentru comunicarea și căutarea la distanță a informațiilor.

Toate aceste aspecte au fost și sunt într-o interdependență continuă ținând seama de particularitatea informaticii care oferă sisteme de calcul performante și produse *software* competitive în rezolvarea problemelor. Utilizarea efi-

cientă a sistemelor de calcul și a produselor *software* necesită o instruire continuă, atât pentru programatori, cât și pentru utilizatori.

Dezvoltarea gândirii algoritmice trebuie luată în considerare ca obiectiv în instruire, atunci când se învață *algoritmica* (metode și tehnici) dar și când se învață *programarea* (utilizarea limbajelor de programare).

Practica instruirii elevilor și studenților a demonstrat că învățarea unui limbaj de programare este în general "mai ușoară" decât învățarea elaborării algoritmilor (algoritmica). Acest lucru se poate justifica prin faptul că elaborarea unui algoritm este echivalentă cu implementarea (reprezentarea) raționamentelor (proceselor demonstrative) deduse din metodele și tehnicile utilizate în rezolvarea unei probleme.

Rezolvarea problemelor necesită nu numai cunoștințe clare și precise, dar și capacitate de sinteză și control și mai ales capacitate de creație. Dacă vrem să facem o analogie, un ***programator*** poate fi "*compozitorul*" care realizează o "*lucrare muzicală*".

În toate etapele instruirii trebuie avută în vedere următoarea interdependență: "***sistem de calcul - algoritmică - programare***".

Succesele unui concurent la olimpiadele de informatică depind de cunoașterea utilizării unui limbaj de programare modern, dar mai ales depind de "bogăția" și stăpânirea cunoștințelor în elaborarea algoritmilor. Și mai există încă ceva: experința acumulată în activitatea de rezolvare a problemelor prin formarea unei gândiri algoritmice solide și consistente. Același lucru se poate afirma și despre succesele unui programator care lucrează într-o firmă de *software*.

Complexitatea problemelor care necesită descrierea mai multor procese de calcul complexe a determinat folosirea noțiunii de *algoritm* în activitatea de rezolvare a problemelor. Multe procese naturale, multe activități umane, pot fi descrise într-o *formă algoritmică* prin definirea unor *informații* și *acțiuni* clare și precise, eliminându-se ambiguitățile în *interpretare* și în *operații*. Algoritmizarea este o cerință fundamentală în rezolvarea oricărei probleme cu ajutorul calculatorului.

Experiența a demonstrat că nu orice problemă poate fi rezolvată prin algoritmizarea rezolvării, adică prin descrierea unui algoritm de rezolvare. Așa s-a delimitat *clasa problemelor decidabile* (o problemă este ***decidabilă*** dacă există *un algoritm* pentru rezolvarea ei) de *clasa problemelor nedecidabile* (o problemă este ***nedecidabilă*** dacă nu există un *algoritm* pentru rezolvarea ei).

Un algoritm implementează diverse metode și tehnici de rezolvare care au fost descoperite sau definitivate într-un anumit moment în evoluția științifică a domeniului respectiv.

Există algoritmi care urmează metode dezvoltate înainte de apariția calculatoarelor, dar cele mai multe probleme cer abordări noi. Chiar dacă ne gândim numai la "problema celor patru culori", care a fost rezolvată (în anul 1977) doar prin utilizarea calculatorului și prin utilizarea unei metode "noi" (metoda ***backtracking***), facem dovada acestei afirmații.

Gândirea algoritmică

În etapa actuală de dezvoltare științifică și tehnică, rezolvarea unei probleme dintr-un domeniu (informatică, matematică, fizică, chimie etc.) reprezintă *o activitate de creație*, *un raționament* prin *construirea*, *generarea*, *descrierea* unui:

- ***proces demonstrativ*** (*demonstrația*) care să arate *existența* uneia sau mai multor soluții și / sau să determine efectiv soluțiile exacte;
- ***proces computațional*** (*algoritmul*) care să codifice un *proces demonstrativ*, o metodă sau o tehnică de rezolvare în scopul determinării (eventual aproximative) a soluțiilor exacte.

În general, în procesul (activitatea) de rezolvare a unei probleme dintr-un anumit domeniu (științific, economic, social etc.), este necesară evidențierea *ipotezei* (condițiile, stările inițiale, parametri inițiali) și a *concluziei* (cerințele, obiectivele, scopurile) din analiza și studiul enunțului problemei.

Procesul de rezolvare (raționamentul) constă în utilizarea selectivă a *legilor*, *teoremelor*, *propozițiilor* etc. din domeniul problemei, pentru ca pornind de la *ipoteză*, prin aplicarea succesivă a *legilor*, *teoremelor* etc. să se obțină *concluzia* problemei.

Legătura dintre *ipoteză*, *concluzie* și *procesul de rezolvare* (raționamentul) determină o structură asemănătoare conceptului de program, și anume (*Ipoteză - Date de intrare; Concluzie - Date de ieșire*):

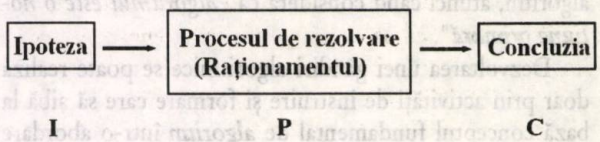


Figura 1

De exemplu, procesul de rezolvare a unei probleme poate fi descris astfel: există un număr de teoreme $T_1, ..., T_n$ determinate de ipotezele $I_1, ..., I_n$ și de concluziile $C_1, ..., C_n$; acestea sunt selectate și apoi aplicate astfel încât să se poată realiza identificările : $I \supseteq I_1$, $C_1 \supseteq I_2$, ..., $C_n \supseteq C$, adică:

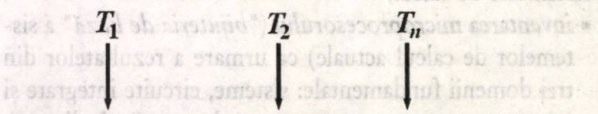


Figura 2

Această prezentare poate fi înțeleasă dacă, de exemplu, facem analogia cu rezolvarea problemelor de geometrie

care necesită delimitarea *ipotezei* și a *concluziei*, ca apoi să se utilizeze numai acele *teoreme, propoziții, proprietăți* care, pornind de la *ipoteză*, prin aplicarea lor succesivă, să se obțină *concluzia* cerută.

Evident, din experința căpătată în rezolvarea problemelor, se poate afirma faptul că selectarea teoremelor și aplicarea lor se poate realiza doar prin "stăpânirea" domeniului respectiv la nivel de specialist sau de expert.

Este cunoscut faptul că ființele umane pot rezolva o problemă în trei moduri (metode evidențiate de inteligența artificială):

- pornind de la *ipoteză* și obținând *concluzia* ("*metoda înlănțuirii înainte*");
- pornind de la *concluzie* și obținând *ipoteza* ("*metoda înlănțuirii înapoi*");
- pornind simultan, de la *ipoteză* și *concluzie* ("*metoda mixtă*").

În toate științele (matematică, fizică, chimie etc.) există multe probleme care, chiar dacă au fost studiate într-o anumită etapă, abordarea lor cu metode și tehnici moderne poate să trezească oricând interesul tinerilor, al cercetătorilor în general. Această afirmație poate fi susținută cu atât mai mult cu cât, astăzi, prin utilizarea unor metode specifice tehnologiei informației și prin utilizarea calculatorului, se pot concepe tehnici care altă dată erau imposibil de imaginat.

Din punct de vedere metodologic, trebuie să ne obișnuim să reformulăm problemele în mod explicit și adecvat rezolvării lor matematice sau - când este posibil - cu ajutorul calculatorului. Pentru acest lucru trebuie să cunoaștem "*limitele*" gândirii demonstrative și în același timp "*limitele*" gândirii algoritmice, apoi să cunoaștem performanțele calculatoarelor.

Să amintim aici istoria problemei celor patru culori care începe în anul 1852, când studentul englez ***Francis Guthrie*** a enunțat următoarea problemă pentru colorarea unei hărți [11, 21]:

"*Sunt suficiente patru culori pentru a colora o hartă care reprezintă diverse țări, cu condiția ca oricare două țări vecine (care au frontiera comună) să fie colorate cu culori diferite*".

Mulți matematicieni cu renume au studiat această problemă (***A. de Morgan, A. Cayley, A. B. Kempe, H. Heesch, K. Appel, W. Hakel***), dar rezolvarea ei completă nu a fost posibilă decât în anul 1977 (***K. Appel, W. Hakel***, "*The four color problem*" în Mathematics Today, L. A. Steen(ed.), Springer Verlag, 1978) și acest lucru numai după ce s-a utilizat calculatorul, deoarece fiind o problemă combinatorială cu spațiul soluțiilor de ordin foarte mare, a fost necesară conceperea unei metode algoritmice pentru căutarea eficientă cu ajutorul calculatorului.

Teoretic, problema a fost rezolvată în cazul a cinci culori. De altfel, rezolvarea acestei probleme a impus în informatică o metodă devenită clasică în domeniul elaborării algoritmilor: metoda ***backtracking***.

În zilele noastre apar tot mai des probleme a căror rezolvare (*proces demonstrativ*) este rezultatul îmbinării metodelor pur matematice cu metodele algoritmice (computaționale).

Uneori, rezultatele computaționale pot oferi surse de inspirație pentru metodele matematice. Informaticienii au și venit în întâmpinarea matematicienilor și specialiștilor prin elaborarea diverselor sisteme de programe care realizează calcule numerice, logice, simbolice, reprezentări spațiale etc. cu ajutorul cărora pot fi dobândite cunoștințe de matematică sau se pot rezolva probleme din diverse domenii.

Totuși, aceste aspecte nu trebuie să conducă la neglijarea procesului demonstrativ. Cele mai utilizate astfel de sisteme de programe sunt: ***Mathematica***, ***Statistica***, ***Matlab***, ***Mathcad***, ***Derive***, ***Maple***, ***Origin***, ***SlideWrite***, ***Workplace***, ***Eureka*** etc.

Este cunoscut faptul că în științele naturii se obțin rezultate deosebite prin îmbinarea metodelor teoretice cu metodele experimentale. Pentru matematică (și alte științe), informatica (tehnologia informației) este cea care oferă rezultate experimentale prin utilizarea calculatorului în rezolvarea problemelor.

Metoda algoritmică va fi un experiment pentru justificarea rezultatelor procesului demonstrativ de la metoda matematică, dar în același timp poate să fie o metodă de rezolvare independentă.

Metoda algoritmică trebuie să substituie rezultatele obținute pe cale teoretică prin metode computaționale eficiente ținând seama de limitele proceselor algoritmice și de performanțele calculatoarelor. Este adevărat că uneori metodele matematice pot conduce la simplificarea metodelor algoritmice și invers.

Elevii care participă la olimpiadele școlare de informatică au constatat că, de multe ori, o analiză profundă a abordării matematice poate să conducă la obținerea unei metode algoritmice eficiente.

Participanții la olimpiadele de matematică pot constata același lucru la o analiză profundă a abordării algoritmice. De aici, importanța atât a proceselor demonstrative (metode matematice), cât și a proceselor computaționale (metode algoritmice).

În funcție de natura metodelor / tehnicilor implementate în procesele computaționale, algoritmii pot fi: numerici, seminumerici, formali, combinatoriali, neuronali, de căutare, de sortare, recursivi, de rescriere, secvențiali, paraleli, determiniști, nedeterminiști, probabilistici, aleatori, euristici, de tip *Monte Carlo*, genetici, de simulare, geometrici computaționali etc.

Definiții matematice

Al-Khwarizmi, Muhammed ibn Musa (cca. 780-cca. 850) a folosit pentru prima dată reguli precise și clare pentru a descrie procese de calcul (operații aritmetice fundamentale)

în lucrarea sa "*Scurtă carte despre calcul algebric*".

Figura 3

Mai târziu, această descriere apare sub denumirea de algoritm în "*Elementele lui Euclid*". Algoritmul lui *Euclid* pentru calculul celui mai mare divizor comun a două numere naturale este, se pare, primul algoritm cunoscut în matematică.

Inițial, noțiunea de algoritm a fost considerată primară. În matematica modernă noțiunea de algoritm a primit mai multe definiții:

- algoritmul normal al lui *A. A. Markov*;
- algoritmul operațional al lui *A. A. Leapunov*;
- mașina *Turing* / mașina cu stivă;
- funcții recursive;
- sisteme *POST*.

Prima descriere a definiției matematice pentru noțiunea de algoritm a fost dată de matematicianul rus *A. A. Markov*. S-a demonstrat că din punct de vedere matematic aceste definiții sunt echivalente.

În prezent, în domeniul probabilităților și statisticii, sunt cunoscute așa-numitele *procese Markov*. De altfel, astăzi se întâlnește foarte frecvent conceptul de proces sub diverse ipostaze: în reprezentarea și execuția algoritmilor, în funcționarea sistemelor de calcul, în execuția sistemelor de operare, în execuția programelor, în funcționarea rețelelor de calculatoare etc.

În [5] se precizează: "*... vom înțelege prin problemă algoritmică o funcție total definită P: $I \to F$, unde I este mulțimea datelor de intrare ale algoritmului, iar F este mulțimea informațiilor finale. Vom presupune că I și F sunt nevide și cel mult numărabile. O instanță a problemei P este problema P(i), pentru o valoare particulară i din I. Intuitiv, considerăm că algoritmul este o pereche de funcții (predicate) a = (f, g), unde f: $I \to [0,1]$, g: $I \times F \to [0, 1]$. Predicatul f descrie elementele lui I care pot fi folosite ca întrebări, iar predicatul g indică legăturile care trebuie să existe între valorile de intrare și cele de ieșire la terminarea algoritmului pentru care rezultatele furnizate de algoritm să fie corecte.*"

Ținând seama de o astfel de definiție, algoritmul (f, g) este parțial corect pentru problema P, dacă pentru orice instanță $P(i)$ este adevarată în ipoteza că $g(I, p(I))$ există. De asemenea, algoritmul (f, g) este total corect pentru problema P, dacă pentru orice instanță $P(i)$, cu $f(i)$ adevărată, $g(i, P(i))$ există și este adevarată.

În [6] se consideră că "*Un algoritm poate fi definit ca o funcție f: $D \to F$, unde F este mulțimea informațiilor finale, iar D este mulțimea informațiilor inițiale*".

Se evidențiază faptul că un algoritm utilizează o memorie virtuală și că prin executarea instrucțiunilor algoritmului se realizează schimbări ale valorilor unor locații de memorie, astfel că starea celulelor generează o succesiune de stări ale algoritmului (în execuție, algoritmul este un proces dinamic).

O imagine intuitivă asupra noțiunii de algoritm este definiția următoare: se presupun n celule de memorie $c_1, c_2, \ldots, c_n$, în care se pot introduce numere sau expresii, asfel încât fiecare celulă conține, la un moment dat, un număr sau o expresie ori poate să fie vidă. Celulele c_i, $i = 1, 2, \ldots, n$, împreună cu informația conținută în ele formează starea inițială S_0. În funcție de aceasta, algoritmul f efectuează anumite modificări asupra conținutului celulelor c_i. Se utilizează asfel o stare intermediară S_1. Pornind de la S_1, algoritmul f asigură în același mod, trecerea la o stare S_2 etc. Un sistem de reguli al algoritmului precizează care stare, obținută la un moment dat, poate fi declarată stare finală S_k: $S_0 \to S_1 \to \ldots \to S_{k-1} \to S_k$.

Vom vedea în cele ce urmează că pentru execuția unui algoritm este necesară o mașina virtuală pentru interpretarea și execuția instrucțiunilor descrise de algoritm. Această execuție constă în generarea și desfășurarea unui proces dinamic care va utiliza o memorie virtuală și un procesor virtual pentru realizarea operațiilor invocate de execuția instrucțiunilor.

De fapt, procesele care se generează într-un sistem de calcul prin execuția unui program scris într-un limbaj de programare, sunt reprezentări efective și reale ale proceselor care se generează prin execuția unui algoritm pe o mașină virtuală; mașina virtuală trebuie să simuleze o mașină reală, un calculator și trebuie să fie constituită din memorie virtuală, procesor virtual și dispozitive de intrare / ieșire virtuale).

Definiții informatice

"*Prin algoritm se înțelege o prescripție care determină un proces de calcul și care este precisă, perfect inteligibilă, nepermițând nici un fel de interpretări din partea celui care o duce la îndeplinire*"[8].

În [2] noțiunea de algoritm se definește astfel: "*Algoritmul desemnează o mulțime exhaustivă și univoc determinată de operații, împreună cu succesiunea în care trebuie aplicate asupra datelor inițiale ale problemei, pentru a obține soluția. El este considerat mod de prezentare a procesului de rezolvare a problemelor și suport după care se realizează programul*".

De asemenea, în [7] se precizează următoarele: "*Algoritmul este o noțiune primară. De obicei, prin algoritm se înțelege o secvență finită și ordonată de operații, care pornind de la o mulțime finită de date (inițiale), conduce (prin aplicarea în mod mecanic și uneori repetată a operațiilor) la o mulțime finită de rezultate. [...] Modul de rezolvare a unei probleme este un algoritm, căci pornind de la date (prezentate în enunțul problemei) și aplicând acestora un număr finit de operații ordonate (transformări, rationa-*

mente etc.), ajungând la soluția problemei, adică la rezultate".

În [1] conceptul de algoritm este descris astfel: "*Prin algoritm vom înțelege o secvență finită de comenzi explicite și neambigue care executate pentru o mulțime de date (ce satisfac anumite condiții inițiale I), conduce în timp finit la rezultatul corespunzător.*"

De asemenea, pentru înțelegerea noțiunii de algoritm, în [2] se precizează: "*Soluția unei probleme, din punct de vedere informatic, este dată printr-o mulțime de comenzi (instrucțiuni) explicite și neambigue, exprimate într-un limbaj de programare. Această mulțime de instrucțiuni prezentată conform anumitor reguli sintactice formează un program. Un program poate fi privit și ca un algoritm exprimat într-un limbaj de programare. Totuși, un algoritm descrie soluția problemei independent de limbajul de programare în care este redactat programul*".

În legătură cu rezolvarea problemelor sub formă algoritmică, în [2] se arată faptul că nu este suficient ca o anumită problemă (clasă de probleme) să admită soluții (chiar și o singură soluție). Din punct de vedere al informaticii (al practicii), interesează dacă există soluția care poate fi descrisă algoritmic, deci dacă soluția problemei poate fi construită efectiv. De asemenea, pentru rezolvarea anumitor probleme pot exista mai mulți algoritmi (de exemplu, problemele de sortare [23]).

Stabilirea celui mai bun dintre aceștia se realizează în urma unui proces de analiză prin care se determină performanța fiecăruia [1, 5, 12, 24, 25, 26].

Principalele caracteristici ale unui algoritm trebuie să fie ([1, 12, 25]):

- ***generalitate*** - algoritmul trebuie să fie cât mai general astfel ca să rezolve o clasă cât mai largă de probleme;
- ***claritate / determinare*** - acțiunile algoritmului trebuie să fie clare, simple și riguros specificate;
- ***finitudine*** - acțiunile algoritmului trebuie să se termine după un număr finit de operații, aceasta pentru orice set de date valide;
- ***corectitudine*** - algoritmul trebuie să producă un rezultat corect (date de ieșire) pentru orice set de date de intrare valide;
- ***performanță*** - algoritmul trebuie să fie eficient privind resursele folosite, și anume să utilizeze memorie minimă și să se termine într-un timp minim.

Întreaga activitate de cercetare și elaborare de *software* din domeniul tehnologiei informației este determinată de inventarea, conceperea, elaborarea, testarea și implementarea de algoritmi performanți și utili.

Marea diversitate a algoritmilor și marea aplicabilitate a acestora în toate domeniile, face ca această temă să fie mereu actuală și într-o continuă schimbare și perfecționare.

Bibliografie

1. Albeanu Gr., *Algoritmi și limbaje de programare*, Universitatea "Spiru Haret", Editura România de Mâine, București, 2000
2. Apostol C., Roșca I. Gh., Roșca V., Ghilic-Micu B., *Introducere în programare. Teorie și aplicații*, Editura București, 1993
3. Georgescu H., *Programare concurentă. Teorie și aplicații*, Editura Tehnică, București, 1996
4. Kinston J. H., *Algorithms and Structures Design. Corectness, Analysis*, Addison Wesley, Longman, 1998
5. Mitrana V., *Provocarea algoritmilor. Probleme pentru concursurile de informatică*, Editura Agni, București, 1994
6. Odăgescu I., *Optimizarea algoritmilor*, Editura Militară, Bucuresti, 1991
7. Perjeriu E., Văduva I., *Îndrumar pentru lucrări de laborator la cursul de Bazele Informaticii*, Universitatea din București, 1986
8. Popovici C., Georgescu H., State L., *Bazele Informaticii*, vol. I, Universitatea din București, 1990
9. Skiena S., *The Algorithm Design Manual*, Springer Verlag, New York, Inc., 1998
10. Vlada M., *Informatica*, Universitatea din București, Editura Ars Docendi, București, 1999
11. Vlada M., *Poligoane stelate. Problema lui Hopf și Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348
12. Zaharia M. D., *Structuri de date și algoritmi. Exemple în limbajele C și C++*, Editura Albastră, Cluj-Napoca, 2002
13. Cristea V., Giumale C., Kalisz E., Păunoiu Al., *Limbajul C standard*, Editura Teora, București, 1992
14. Popovici M. D., Popovici M. I., *C++. Tehnologia orientată spre obiecte. Aplicații*, Editura Teora, București, 2000
15. www.math.gatech.edu/~thomas/FC/fourcolor.html
16. mathworld.wolfram.com/Four-ColorProblem.html
17. spicerack.sr.unh.edu/~student/tutorial/fourColor/FourColor.html
18. mathcentral.uregina.ca/RR/database/RR.09.97/fisher1.html
19. www.uccs.edu/~asoifer/book5.html
20. www.uccs.edu/~asoifer/solve98.html
21. www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg
22. mathworld.wolfram.com/Algorithm.html
23. www.cs.rit.edu/~atk/Java/Sorting/sorting.html
24. faculty.juniata.edu/rhodes/cs2/ch129.htm
25. www.scism.sbk.ac.uk/law/Section5/chap1/ s5c1p2.htm
26. www.ics.uci.edu/~eppstein/161/people.html
27. www.cs.pitt.edu/~kirk/algorithmcourses/
28. www.gnacademy.org/text/cc/
29. java.sun.com/docs/books/tutorial/java/concepts/
30. www.accu.org/acornsig/public/articles/oop_c.html
31. loki.cs.brown.edu:8081/webae/full.html

Domnul conf. dr. Marin Vlada este cadru didactic la Universitatea București și poate fi contactat prin e-mail la adresa vlada@chem.unibuc.ro.

COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui episod al serialului dedicat compresiilor de date vă vom prezenta compresia aritmetică, cea mai eficientă metodă entropică de compresie.

În articolul din numărul anterior am prezentat o serie de algoritmi pentru compresia datelor care foloseau *arbori Huffman*. Dezavantajul acestor algoritmi este acela că fiecare simbol generat de o sursă de informație S este codificat folosind un număr întreg de biți, fapt care duce la apariția unei diferențe mari între lungimea codificării unui șir de simboluri și entropia acestuia, dacă nu se consideră lungimea, în biți, a dicționarului.

Nu există algoritmi care să elimine complet redundața unei surse de informație deoarece, în primul rând, entropia unui șir de simboluri generat de o sursă este un număr real și bitul este o unitate atomică, și, în al doilea rând, trebuie transmis dicționarul pentru ca informația să poată fi reconstituită.

Cea mai eficientă metodă entropică de compresie a datelor care elimină aproape complet redundanța unei surse de informație este ***compresia aritmetică.***

Algoritmul de compresie

Compresia aritmetică a fost descoperită de către cercetătorii *Peter Elias*, *Jorma J. Rissanen* și *Richard C. Pasco*. Ideea care stă la baza acestei metode de compresie este aceea de codifica un șir de simboluri folosind un singur cod și nu câte un cod pentru fiecare simbol în parte. De fapt, orice șir de simboluri generat de o sursă de informație este codificat folosind un număr real cuprins în intervalul [0, 1).

Datorită faptului că această metodă de compresie este entropică avem nevoie de probabilitățile p_i ($0 \le i < m$, unde m este numărul de simboluri pe care le poate genera o sursă de informație S) de apariție ale simbolurilor.

Similar algoritmului *Huffman* se pot construi variantele *statică*, *semi-statică* și *dinamică* ale algoritmului de compresie aritmetică.

În continuare vom prezenta modul în care se realizează codificarea unui șir de simboluri generat de o sursă de informație S pentru varianta *statică*. În cazul variantei *semi-statice* probabilitățile se calculează pe baza simbolurilor generate de sursa de informație S, fiind nevoie de două parcurgeri ale șirului de simboluri la fel ca în cazul algoritmului *Huffman*.

Considerăm un interval $[a, b) \subseteq [0, 1)$, $b > a$. Fiecărui simbol care poate fi generat de o sursă de informație S i se atașează un subinterval al lui $[a, b)$ cu proprietatea că lungimea subintervalului corespunzător unui simbol este direct proporțională cu probabilitatea de apariție a simbolului respectiv și oricare două subintervale corespunzătoare la două simboluri distincte nu au puncte de intersecție.

Fie P_i probabilitatea cumulată a simbolului A_i, $P_0 = 0$ și $P_i = p_0 + ... + p_{i-1}$, $0 \le i < m$, și fie $l = b - a$ lungimea intervalului $[a, b)$. Din faptul că suma tuturor probabilităților de apariție ale simbolurilor A_i, $0 \le i < m$ este 1, rezultă că $P_i \in [0, 1)$.

Fiecărui simbol A_i i se va atașa intervalul $[a + l \cdot P_i, a + l \cdot (P_i + p_i))$. Lungimea subintervalului corespuzător unui simbol A_i este egală cu $l \cdot p_i$.

În cazul în care o sursă de informație nu generează toate simbolurile pe care le poate genera, atunci probabilitatea de apariție a unor simboluri poate fi 0 deci, lungimea subintervalelor atașate simbolurilor care nu sunt generate este 0, caz în care avem intervale degenerate, și, dacă p_i este 0 și p_{i+1} este diferit de 0, atunci intersecția dintre subintervalele corespunzătoare celor două simboluri este diferită de mulțimea vidă și nu mai sunt respectate condițiile enunțate mai sus. Dacă se elimină subintervalele de lungime 0, atunci condițiile sunt respectate.

Algoritmul de compresie aritmetică se bazează pe această împărțire a unui interval.

Faptul că avem intervale degenerate nu va influența funcționalitatea algoritmului, deoarece o sursă de informație nu va genera niciodată simboluri care au probabilitatea de apariție 0.

Algoritmul de codificare constă în alegerea unui subinterval al intervalului [0, 1) corespunzător primului simbol generat de o sursă S și apoi, ca nou interval se consideră

subintervalul ales și se alege subintervalul corespunzător celui de-al doilea simbol.

Algoritmul de compresie aritmetică folosit pentru a codifica un șir de simboluri generat de o sursă de informație S este următorul:

- considerăm intervalul care are extremitatea din stânga a și lungimea l;
- fie p_i probabilitățile de apariție ale simbolurilor care pot fi generate de o sursă de informație S;
- fie P_i probabilitățile cumulate ale simbolurilor;
- pentru fiecare simbol A_i generat de sursa de informație S execută:
  - $a \leftarrow a + l \cdot P_i$;
  - $l \leftarrow l \cdot p_i$;
- *rezultat* $\leftarrow a + l/2$.

În ciclul repetitiv prezentat anterior se schimbă intervalul inițial cu subintervalul corespunzător simbolului generat de sursa de informație S.

Datorită faptului că intervalele atașate simbolurilor care au probabilitatea de apariție diferită de 0 sunt disjuncte, fiecare simbol este unic determinat de orice număr care aparține subintervalului corespunzător.

În teorie se folosește ca interval inițial intervalul [0, 1), deci cu capătul din stânga 0 și lungimea 1.

După parcurgerea șirului de simboluri generat de o sursă de informație se transmite un număr real din intervalul [0, 1) care reprezintă codificarea șirului de simboluri. Numărul real trebuie transmis cu o precizie foarte mare. Acest număr este dat, de obicei, de mijlocul ultimului interval calculat cu ajutorul algoritmului prezentat anterior: $a + l/2$.

Algoritmul de codificare mai poate fi construit folosind limitele inferioară și superioară ale unui interval în locul limitei inferioare și lungimii intervalului. În acest caz, după linia în care se calculează lungimea intervalului se adaugă linia $b \leftarrow a + l$.

Dacă intervalul inițial este [0, 1), atunci precizia cu care trebuie calculată limita din stânga pentru un simbol cu probabilitatea p_i, $0 \le i < m$, este de $\lceil -\log_2 p_i \rceil$ biți.

Dacă analizăm modul de construire al numărului real plecând de la intervalul [0; 1), atunci precizia cu care trebuie transmis numărul care reprezintă mijlocul ultimului interval calculat este de $\left\lceil n \cdot \left(-\sum_{i=0}^{m-1} p_i \cdot \log_2 p_i \right) \right\rceil + 1$ biți, unde n reprezintă numărul de simboluri generate de sursa S.

Se poate observa foarte ușor că diferența dintre entropia șirului generat de sursa de informație S și numărul mediu de biți necesari transmiterii unui simbol este foarte mică (mai mică de 1 bit).

De exemplu, fie o sursă de informație S care poate genera simbolurile `'a'`, `'b'` și `'c'` cu probabilitățile 1/2, 1/4 și 1/4.

În figura 1 se poate observa modul de codificare a șirului de simboluri `'abac'`. Precizia cu care trebuie transmis rezultatul este de 7 biți (rezultatul este 0,3046875 = $0{,}0100111_2$ ceea ce înseamnă că se transmit biții 0100111,

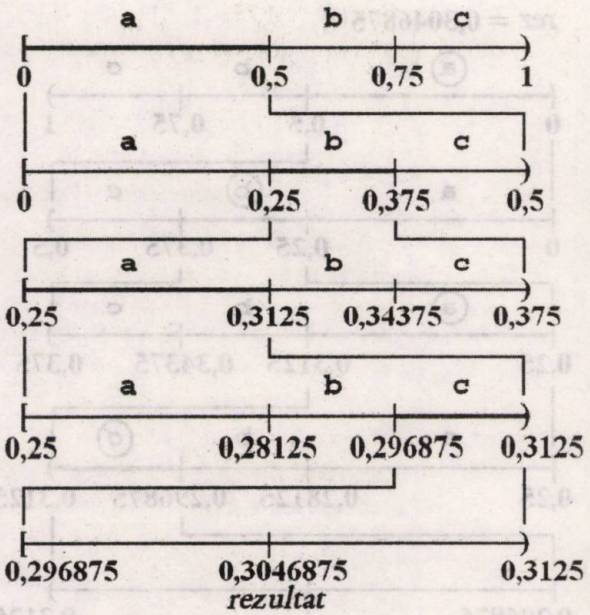


Figura 1

deoarece numărul este cuprins între 0 și 1 și ne interesează doar porțiunea care se află după virgulă).

Pe lângă rezultatul obținut în urma codificării mai trebuie transmis *dicționarul* care este format din probabilitățile de apariție ale simbolurilor care pot fi generate de sursa de informație S.

Motivul pentru care rezultatul îl reprezintă mijlocul ultimului interval calculat, și deci o creștere a preciziei cu un bit, este acela că în practică există situații în care reprezentarea în baza 2 a celor două limite ale intervalului, folosind aceeași precizie, sunt egale pentru că microprocesoarele existente realizează operații cu numere reale cu o precizie finită de până la 80 de biți.

Din punct de vedere teoretic este suficient ca rezultatul să fie constituit de limita inferioară a ultimului interval găsit, calculată cu precizia de $\left\lceil n \cdot \left(-\sum_{i=0}^{m-1} p_i \cdot \log_2 p_i \right) \right\rceil$ biți, deoarece, bazându-ne pe cele enunțate anterior, un simbol căruia i s-a atașat intervalul [a, b) este unic determinat de orice număr real care aparține intervalului.

Se poate pune întrebarea "*De ce nu se poate lua ca rezultat limita superioară a intervalului corespunzător ultimului simbol codificat*?".

Răspunsul este acela că limita superioară a intervalului corespunzător ultimului simbol codificat nu aparține intervalului, aceasta aparține intervalului corespunzător simbolului care urmează în ordine lexicografică.

Algoritmul de decompresie

Pentru a decodifica un număr real, cuprins între 0 și 1, a cărui lungime în biți se cunoaște, trebuie să avem probabilitățile de apariție ale simbolurilor care au fost folosite în procesul de codificare și numărul total n al simbolurilor care au fost codificate.

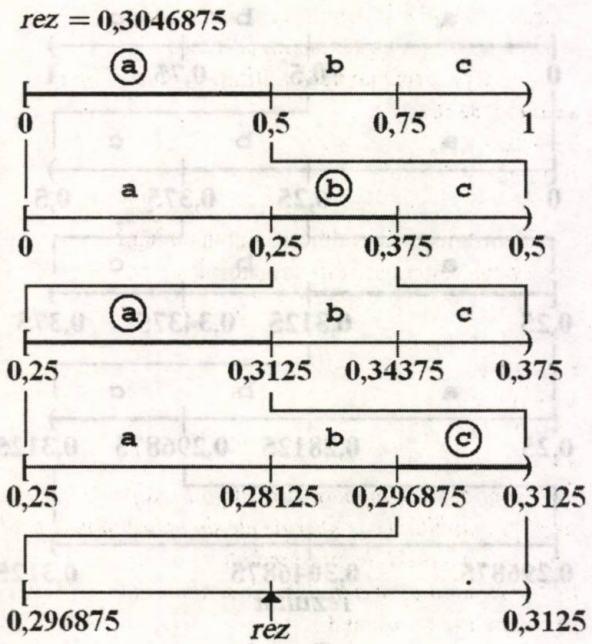


Figura 2

La începutul procesului de decodificare considerăm intervalul $[a, a + l)$, unde $a = 0$ și $l = 1$. Fiecărui simbol îi corespunde un subinterval al acestui interval.

În continuare se caută subintervalul căruia îi aparține numărul care trebuie decodificat. După ce s-a găsit acest subinterval se transmite simbolul corespunzător acestuia și noul interval devine cel găsit. Acest pas se execută până în momentul în care am decodificat n simboluri.

Dacă nu se transmite numărul n de simboluri care au fost codificate, în momentul compresiei alfabetul sursei S de informație se poate extinde cu un simbol suplimentar care are semnificația de *sfârșitul codificării* care se va codifica după ce sursa S nu mai generează simboluri. În concluzie, în momentul în care numărul se va afla în subintervalul corespunzător simbolului de *sfârșit de codificare*, decodificarea se va încheia.

În figura 2 se poate observa modul de decodificare a numărului 0,3046875 pentru alfabetul format din simbolurile `'a'`, `'b'` și `'c'`, care au probabilitățile 1/2, 1/4 și 1/4, și numărul de simboluri care au fost codificate $n = 4$.

La primul pas se observă că numărul ***rez*** aparține subintervalului [0, 0,5), subinterval corespunzător simbolului `'a'`, deci acest simbol se va transmite și noul interval va deveni [0, 0,5).

La al doilea pas numărul ***rez*** aparține subintervalului [0,25, 0,375) care corespunde simbolului `'b'`, acest simbol se va transmite și noul interval este [0,25, 0,375).

La al treilea pas numărul ***rez*** aparține subintervalului [0,25, 0,3125) care corespunde simbolului `'a'`, acest simbol se va transmite și noul interval este [0,25, 0,3125).

La al patrulea pas numărul ***rez*** aparține subintervalului [0,296875, 0,3125) care corespunde simbolului `'c'`, acest simbol se va transmite și noul interval este [0,296875, 0,3125). În acest moment se încheie procesul de decodificare deoarece au fost decodificate $n = 4$ simboluri. În timpul decodificării a fost transmis șirul de simboluri `'abac'`.

Detalii de implementare

Datorită faptului că pentru date de dimensiuni mari precizia cu care trebuie transmis rezultatul devine foarte mare, calculatoarele nu dețin resurse suficiente pentru a efectua operații cu numere reale cu precizie mare. De exemplu, există situații când trebuie folosită o precizie de $8 \cdot 10^5$ biți sau chiar mai mare, în timp ce calculatoarele moderne au o precizie de cel mult 128 de biți. În acest caz, algoritmii prezentați anterior nu se pot utiliza în practică.

Toate implementările performante ale acestui algoritm nu utilizează operații cu numere reale, ci folosesc operații cu numere întregi reprezentate pe 16 sau 32 de biți pentru a evita erorile de calcul care apar când se efectuează operații cu numere reale reprezentate în virgulă mobilă. În unele implementări, cele două operații de înmulțire, care apar în algoritmii prezentați, au fost înlocuite cu adunări și deplasări pe biți. Cu toate că s-a obținut o scădere a timpului de execuție pentru date de dimensiuni mari, acești algoritmi nu mai codifică informația la fel de eficient.

Pentru a implementa acest algoritm fără a folosi numere reale vom utiliza limitele superioare și inferioare ale intervalelor în locul limitelor inferioare și lungimilor intervalelor.

Datorită faptului că pentru date de dimensiuni mari nu putem păstra în memorie tot rezultatul codificării, ajungem la concluzia că acesta trebuie transmis pe parcursul codificării simbolurilor generate de sursa de informație.

În primul rând, având în vedere că se folosesc numere întregi fără semn, trebuie să alegem o reprezentare pentru intervalul [0, 1). În cazul în care folosim întregi pe 16 biți, este de preferat ca acest interval să fie reprezentat prin intervalul $[0, 10000_{16}]$ din care vom considera numai numerele naturale, adică toate numerele întregi fără semn din acest interval $(0, 1, ..., 65535 = FFFF_{16})$.

Vom considera că un număr întreg reprezentat pe 16 biți reprezintă valoarea unui număr real cuprins între 0 și 1 reprezentat cu o precizie de 16 biți. În aceste condiții cel mai apropiat număr de 1 reprezentat cu o precizie de 16 biți este $0{,}1111111111111111_2 = 0{,}99998$.

Având reprezentarea intervalului [0, 1) și cunoscând faptul că suma tuturor probabilităților de apariție a simbolurilor generate de o sursă de informație S este 1 (adică lungimea intervalului) aceste probabilități trebuie recalculate astfel încât suma lor să fie 65535, deci vor fi înmulțite cu acest număr. Din punct de vedere teoretic suma probabilităților ar trebui să fie 65536, dar în cazul practic nu se întâmplă acest lucru deoarece este vorba de o mulțime și nu de un interval și între 65535 și 65536 nu mai avem nici un număr. În cazul intervalului [0, 1) există un număr real cu suficient de multe cifre zecimale care să-l aproximeze pe 1.

După efectuarea calculelor exemplificate anterior trebuie avut în vedere faptul că avem la dispoziție un număr limitat de numere din interval și o precizie de maxim 16 biți. Cu aceste valori calculate, dacă se folosește algoritmul de codificare prezentat, în câțiva pași se va ajunge la depășire inferioară (*underflow*) datorită împărțirilor.

Considerând rezultatul codificării unui șir de simboluri $a_1, a_2, ..., a_n$ care au probabilitățile de apariție $p_1, p_2, ..., p_n$ și rezultatul codificării aceluiași șir de simboluri la care s-a mai adaugat un simbol a_j, se poate observa că diferența de lungime dintre cele două rezultate este aproximativ egală cu $\lceil -\log_2 p_j \rceil$, unde p_j este probabilitatea de apariție a simbolului a_j, și toți biții primului rezultat cu excepția ultimilor cel mult 2 biți sunt identici cu biții cei mai semnificativi ai celui de-al doilea rezultat.

Ținând cont de observația anterioară, pe parcursul codificării, în momentul în care cei mai semnificativi biți ai limitelor intervalului de la iterația curentă sunt identici, atunci aceștia se vor transmite și limitele intervalului se vor deplasa la stânga cu un număr de poziții egal cu numărul de biți care au fost identici. Cei mai nesemnificativi biți ai limitei din dreapta a intervalului curent se vor completa, după deplasare, cu 1 pentru ca această limită să aproximeze mai bine limita reală a subintervalului curent.

Datorită faptului că probabilitățile de apariție au fost recalculate și sunt numere întregi, atunci numărul de biți identici ai celor două numere care reprezintă capetele unui interval nu depășesc 16 biți.

Algoritmul de codicare a unui șir de simboluri generate de o sursă de informație *S* folosind aritmetica numerelor întregi fără semn pe 16 biți este următorul:

```
a ← 0
b ← FFFF16
l ← b - a
transmite numărul n de simboluri pe care le va genera sursa S
fie pi probabilitățile de apariție ale simbolurilor care pot fi generate de o sursă de informație S
fie Pi probabilitățile cumulate ale simbolurilor
transmite probabilitățile pi
cât timp sursa S generează simboluri execută
   fie i numărul de ordine al simbolului generat de S
   a ← a + l · Pi
   l ← l · pi
   b ← a + l
   fie c numărul de biți semnificativi identici ai lui a și b
   dacă c > 0 atunci
      transmite cei mai semnificativi c biți ai lui a
      deplasează a și b cu c poziții la stânga
      setează pe 1 cei mai nesemnificativi c biți ai lui b
   sfârșit dacă
sfârșit cât timp
fie c numărul de biți semnificativi identici ai lui a și b
dacă c > 0 atunci
   transmite cei mai semnificativi c biți ai lui a
   deplasează a și b cu c poziții la stânga
   setează pe 1 cei mai nesemnificativi c biți ai lui b
sfârșit dacă
r ← (b - a)/2
transmite r
```

Algoritmul de decodificare a unui număr real, bazat pe observațiile anterioare este următorul:

```
a ← 0
b ← FFFF16
l ← b - a
r ← 0
k ← 15
fie n numărul de simboluri care au fost codificate
fie pi probabilitățile de apariție ale simbolurilor care pot fi generate de o sursă de informație S
fie Pi probabilitățile cumulate ale simbolurilor
cât timp n > 0 execută
   citeste 1 bit
   r ← r + (1 deplasat cu k poziții la stânga)
   k ← k - 1
   dacă există un unic subinterval al intervalului [a, b) care îl include pe r atunci
      fie i numărul de ordine al simbolului corespunzător subintervalului determinat
      transmite simbolul cu numărul de ordine i
      a ← a + l · Pi
      l ← l · pi
      b ← a + l
      fie c numărul de biți semnificativi identici ai lui a și b
      dacă c > 0 atunci
         deplasează a, b și r cu c poziții la stânga
         setează pe 1 cei mai nesemnificativi c biți ai lui b
         k ← k + c
      sfârșit dacă
      n ← n - 1
   sfârșit dacă
sfârșit cât timp
```

În practică se folosesc implementări ale algoritmilor de codificare și decodificare mult mai performante bazate pe folosirea noțiunilor de aritmetică legate de modul de calcul folosind baze de numerație puteri ale lui 2 (de obicei se lucrează în bazele 2^8 sau 2^{16}) în funcție de tipul de calculator și sistem de operare pe care se va folosi programul de compresie / decompresie.

Compresia aritmetică se folosește în combinație cu alte metode și se obține o reducere considerabilă a conținutului unui fișier sau a datelor transmise printr-o rețea de calculatoare (internet, rețea locală etc.).

Această metodă de compresie a datelor nu este foarte răspândită deoarece compania *IBM* deține patente de inventator asupra câtorva algoritmi performanți de compresie aritmetică.

Limba română						Limba engleză					
a	8,18%	î	0,61%	ş	0,64%	a	8,20%	j	0,19%	s	7,06%
ă	2,94%	j	0,17%	t	5,63%	b	1,06%	k	0,39%	t	9,69%
â	0,96%	k	0,14%	ţ	0,81%	c	3,44%	l	4,48%	u	2,87%
b	1,23%	l	4,58%	u	7,39%	d	3,63%	m	2,81%	v	1,24%
c	3,94%	m	3,62%	v	1,28%	e	12,41%	n	7,64%	w	1,35%
d	3,45%	n	6,69%	w	0,08%	f	2,35%	o	7,14%	x	0,21%
e	12,05%	o	4,87%	x	0,88%	g	1,81%	p	2,03%	y	1,89%
f	2,33%	p	2,92%	y	0,09%	h	3,50	q	0,09%	z	0,05%
g	1,25%	q	0,03%	z	0,80%	i	7,68%	r	6,68%		
h	0,10%	r	10,16%								
i	9,74%	s	2,44%								

Tabelul 1

Compresie aritmetică adaptivă

Şirul de simboluri, care urmează a fi codificat, generate de o sursă de informaţie *S* trebuie parcurs de două ori: la prima parcurgere se calculează frecvenţele (probabilităţile) de apariţie ale simbolurilor, iar la a doua parcurgere se codifică şirul. Din această cauză, timpul de execuţie al programelor de compresie este mare datorită faptului că accesul la discurile fizice, reţea etc. este mai lent decât citirea şi prelucrarea datelor care există în memoria calculatorului.

Pentru a evita parcurgerea de două ori a şirului de simboluri generate de sursa *S* şi, deci, scăderea performanţei aplicaţiei de compresie, probabilităţile se pot calcula adaptiv pe măsură ce simbolurile sunt codificate.

Există cel puţin două posibilităţi de a implementa varianta adaptivă a acestui algoritm.

O primă posibilitate este aceea de a iniţializa la început toate probabilităţile de apariţie cu constante astfel încât suma lor să fie 65535 (în cazul implementărilor care folosesc operaţii aritmetice pe 16 biţi) iar pe parcursul codificării, după fiecare pas, acestea trebuie recalculate în funcţie de simbolul care a apărut ultima dată.

O a doua posibilitate este aceea de a extinde alfabetul sursei de informaţie *S* cu un nou simbol, a cărui semnificaţie este *NYT* (*Not Yet Transmitted* - nu a fost transmis încă). La începutul algoritmului, se iniţializează toate probabilităţile de apariţie ale simbolurilor care pot fi generate de sursa *S* cu 0, iar probabilitatea de apariţie a simbolului *NYT* cu 65535. Pe parcursul codificării şirului de simboluri, la apariţia unui simbol a cărui probabilitate de apariţie este 0, se transmite codul parţial al simbolului *NYT* urmat de codul simbolului care a apărut şi apoi se recalculează probabilităţile de apariţie ale simbolurilor în funcţie de simbolul care a apărut. În cazul în care simbolul nu are probabilitatea de apariţie 0, acesta se codifică şi apoi se recalculează probabilităţile.

Pentru recalcularea probabilităţilor de apariţie a simbolurilor generate de o sursă de informaţie *S* se pot folosi diferite modele statistice, în funcţie de informaţia pe care o transmite sursa.

În cazul în care se cunoaşte faptul că sursa de informaţie transmite un text se pot folosi pentru modelul static probabilităţile de apariţie ale simbolurilor din limba în care a fost scris textul respectiv.

De exemplu, dacă un text a fost scris în limba română şi nu s-au folosit diacritice, atunci, probabilitatea de apariţie a simbolului `'a'` va deveni egală cu suma probabilităţilor de apariţie a simbolurilor `'a'`, `'â'` şi `'ă'`.

În cazul în care se poate şti că dacă apare un simbol a_i este foarte probabil să apară un simbol a_j, atunci, pentru a micşora dimensiunea codificării este indicat să se modifice probabilitatea de apariţie a simbolului a_j.

În tabelul 1 se pot observa probabilităţile de apariţie ale caracterelor în limba română şi în limba engleză. Nu am inclus în tabel probabilităţile de apariţie ale simbolurilor speciale `'.'`, `'!'`, `'?'`, `':'`, `';'` etc. deoarece probabilitatea acestora de apariţie este destul de mică şi poate fi calculată adaptiv. Se poate observa faptul că dimensiunea codificării unui singur simbol generat de o sursă de informaţie *S* este cu atât mai mică, cu cât probabilitatea lui de apariţie este mai mare.

Modelele statistice cu performanţe ridicate folosite în cadrul compresiei textelor se bazează, la fiecare pas, pe ultimele *k* simboluri apărute, *k* fiind cuprins, de obicei, între 0 şi 6.

Unii dintre cei mai performanţi algoritmi de compresie a textelor care se bazează pe modele statistice avansate sunt *DMC* (*Dynamic Markov Coder*) şi *PPM* (*Partial Pattern Matching*).

Majoritatea programelor de compresie bazate pe codificarea aritmetică folosesc modelul ***cvasi-aritmetic***. În cadrul acestui model, probabilităţile cumulate se recalculează în momentul în care cea mai mare dintre probabilităţi atinge o valoare prag *Prg*, şi nu după fiecare simbol generat de o sursă de informaţie. Acest lucru duce la o scădere a timpului necesar unui program de compresie pentru a codifica un şir de simboluri

Claudiu Soroiu este redactor al GInfo şi Poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

TEOREME de programare

Clara Ionescu

Sfătuim programatorii începători (dar și pe cei care se consideră "avansați") să respecte aceste modele de rezolvare a diferitelor probleme, considerate clasice și care apar frecvent ca subprobleme în multe probleme de rezolvat. Astfel, vor putea depana mult mai simplu programele scrise, le vor "citi" ușor și peste mult timp, dar le vor înțelege și alții. Nu este doar o recomandare din partea GInfo, ci și a majorității firmelor dezvoltatoare de software...

Selectarea elementului maxim

Fie următoarele enunțuri de probleme!

P15. Într-un spital s-a măsurat temperatura fiecărui bolnav internat. Să se stabilească cea mai mare temperatură măsurată!

P16. Având la dispoziție un tabel cu numele elevilor participanți la un concurs, să se stabilească acel nume cu care va începe tabelul nominal (în ordine alfabetică)!

P17. Într-o clasă se face o statistică, privind veniturile familiilor elevilor. Să se determine venitul minim, existent printre venituri!

Observăm că rezolvările acestor probleme - probabil - vor fi asemănătoare, deoarece în fiecare se cere stabilirea unei valori care este fie cea mai mare, fie cea mai mică în șirul dat. Bineînțeles, pot exista mai multe astfel de valori egale cu valoarea minimă sau maximă. Dar în problemele considerate nu se cere, de exemplu, să numărăm câți bolnavi au acea temperatură maximă și nu ne interesează nici numele acelui bolnav care este cel mai "fierbinte". Subalgoritmul care rezolvă această problemă (în termeni generali) va lucra cu o buclă de tip **pentru**, deoarece fiecare valoare trebuie prelucrată. În pseudocodul următor lucrăm cu entitățile:

```
  n: întreg          {numărul elementelor șirului de prelucrat}
  x: tablou(1..n): tip element   {elementele șirului dat}
  val: tip element               {rezultatul}
subalgoritm Maxim(n,x,val):
  val ← x[1]
  pentru i ← 2, n execută:
    dacă val < x[i] atunci
      val ← x[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Înainte de a trece la rezolvarea problemelor enunțate, să analizăm pseudocodul subalgoritmului. De ce nu am inițializat valoarea maximului (variabila `val`) cu 0, gândindu-ne că sigur va exista un bolnav cu temperatură pozitivă, deci nu există pericolul ca maximul să rămână 0? În rezolvarea problemei **P15** am putea lucra cu această valoare inițială, dar dacă nu ar fi vorba de temperatura unor bolnavi, ci de temperatura aerului în timpul unei luni într-o iarnă geroasă, ar fi posibil să nu avem nici o valoare pozitivă, și astfel maximul ar rămâne 0, păcălindu-ne... Rezultă că se recomandă inițializarea maximului (minimului) cu o valoare existentă sigur în șir, și, deoarece, de regulă, prelucrăm șirurile începând cu primul element, cel mai firesc este ca în inițializarea respectivă să folosim această valoare. Să analizăm oportunitatea de a implementa acest subalgoritm cu o procedură sau cu o funcție. Avem de determinat o singură valoare, deci funcția "ne face cu ochiul". Dacă am folosi o funcție denumită "`val`", în instrucțiunea **if** am avea un apel recursiv nedorit, deci ne-ar mai trebui o variabilă suplimentară, iar la sfârșit am atribui valoarea acesteia identificatorului de funcție. Presupunem că în programul apelant am declarat tipul:

```
type temp = array[1..50] of Real;
procedure P15(n:Byte; T:temp; var max:Real);
var i:Byte;
begin
  max:=T[1];
  for i:= 2 to n do
    if max < T[i] then
        max:=T[i]
end;
```

Având în vedere simplitatea implementării problemelor **P16** și **P17**, nu le vom detalia. În schimb, le vom modifica enunțul, astfel încât să vedem modul de rezolvare dacă se cere și poziția pe care se află minimul cerut. De exem-

plu, în problema P16 se cere "cel mai mic nume". Să ne imaginăm că în problemă nu se cere numele, ci doar indicele în șirul dat al aceluia care va ajunge în tabelul nominal pe prima poziție! Este evident că rezultatul va fi nu o valoare dată în șir (de data aceasta o valoare de tip **string**), ci un număr de ordine. Presupunem că în programul apelant am declarat tipul:

```
type tabel = array[1..30] of string;
procedure P16_1(n:Byte; t:tabel; var ind:Byte);
var i:Byte;
    min:string;
begin
  min := t[1];     {presupunem că min este primul}
  ind := 1;        {dacă min rămâne primul, ind e 1}
  for i := 2 to n do
    if min > t[i] then begin
      min := t[i];              {cel mai mic nume}
      ind := i                  {indicele acestuia}
    end
end;
```

Nu ne-am pus problema cu privire la "care" minim ne interesează, (respectiv, primul găsit, al doilea sau ultimul) deoarece într-un grup de elevi, probabil nu vor exista doi copii cu nume identic. Dar dacă transpunem problema la un șir de cuvinte luate dintr-un text oarecare, atunci este posibil să existe mai multe asemenea valori minime. Observăm că, dacă am utiliza acest subprogram în cazul respectiv, acesta ne-ar furniza indicele *primului* element având valoarea minimă. Dacă ni s-ar cere *ultimul* astfel de element, atunci ar fi suficient să schimbăm operatorul relațional ">" în ">=". Dar se mai poate schimba ceva pentru a scrie un subprogram care să fie optimizat la "sânge". Vom rezolva problema P16, modificând implementarea în acest sens.

```
procedure P16_m(n:Byte; t:tabel; var min:Byte);
var i:Byte;
begin
  min := 1;        {presupunem că min este primul}
  for i := 2 to n do
    if t[min] >= t[i] then
      min := i;
end;
```

Dacă, după apelarea subprogramului vrem să știm și numele elevului (având valoare minimă) vom cere afișarea elementului `t[min]`.

Să analizăm problema P17 într-o variantă modificată, în care ni s-ar cere numele tuturor copiilor care trăiesc în familii având același venit minim. Rezolvarea acestei probleme se realizează în două etape. Mai întâi stabilim minimul în șirul veniturilor, apoi parcurgem din nou întreg șirul și afișăm elementele (numele copiilor) în cazul cărora venitul familiei este egal cu minimul găsit în prima etapă. În acest program trebuie să declarăm tipurile utilizator:

```
type elev = record
       nume : string;
            venit : Real
     end;
   elevi = array[1..30] of elev;
procedure P17_1(n:Byte; e:elevi; var j:Byte;
                                 var rez:elevi);
var i:Byte;
   min:Real;
begin
  min := e[1].venit;
  for i := 2 to n do
    if min > e[i].venit then
      min := e[i].venit;
  j := 0;
  for i := 1 to n do
    if min = e[i].venit then begin
      Inc(j);
      rez[j] := e[i];      {atribuire la nivel}
    end                {de articol}
end;
```

Dacă nu ne place faptul că trebuie să parcurgem de două ori șirul, se poate scrie un subprogram care rezolvă aceeași problemă cu o singură parcurgere a lui:

```
procedure P17_2(n:Byte; e:elevi; var j:Byte;
                                 var rez:elevi);
var i:Byte;
   min:Real;
begin
  min := e[1].venit;
  j := 1;
  rez[j] := e[1];
  for i := 2 to n do
    if min > e[i].venit then begin
    j := 1;
      rez[j] := e[i];
      min := e[i].venit
    end else
    if min = e[i].venit then begin
      Inc(j);
      rez[j] := e[i];
    end
end;
```

Dar astfel am ajuns la o problemă de *selectare a unor elemente dintr-un șir dat pe baza unei proprietăți*. Aici proprietatea a fost simplă: elementele selectate au avut un câmp egal cu minimul din șirul veniturilor. Pe baza observației avem o nouă teoremă de programare.

Selectarea elementelor

Acest model de algoritm pe de o parte seamănă cu *căutarea liniară*, pe de altă parte cu *numărarea*. Aici trebuie să furnizăm pozițiile a mai multor elemente, având o proprietate dată, deci nu este suficient să le numărăm.

```
subalgoritm Selectare_1(n,X,buc,Y):
  buc ← 0
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      buc ← buc + 1
      Y[buc] ← i
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Se observă că pentru păstrarea rezultatului vom avea nevoie de un tablou nou în cazul căruia nu cunoaştem exact câte elemente va cuprinde. Dar, din moment ce nu este exclus ca fiecare element al şirului dat să aibă proprietatea cerută, acesta va fi declarat având atâtea elemente câte are şirul dat. Modelul de mai sus "adună" indicii elementelor având proprietatea P în vectorul Y, şi totodată obţine şi numărul acestora în buc. Dacă nu avem nevoie de elementele respective colecţionte într-un vector nou, deoarece trebuie doar să le furnizăm ca rezultat (de exemplu, afişându-le) algoritmul este şi mai simplu:

```
subalgoritm Selectare_2(n,X):
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      scrie X[i] {sau i, în funcţie de cerinţe}
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Vom da şi a treia variantă a acestui algoritm. În aceasta, considerăm că după selectarea elementelor căutate nu mai avem nevoie de şirul dat. Rezultă că, în loc să folosim un al doilea şir pentru elementele selectate, putem folosi chiar şirul dat. Pseudocodul acestui algoritm este următorul:

```
subalgoritm Selectare_3(n,X,buc,Y):
  buc ← 0
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      buc ← buc + 1
      X[buc] ← X[i]
                {sau i, în funcţie de cerinţe}
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Oare cum rezolvăm problema, dacă se cere păstrarea, nu a elementelor având proprietatea P, ci a acelora care *nu* au această proprietate? Vom nega condiţia din **if**, veţi răspunde! Aşa este. Dar dacă vrem să păstrăm aceste elemente "nemişcate" pe poziţiile lor? În acest caz, putem *anula prezenţa* acestor elemente, suprascriindu-le cu o valoare specială, aleasă în acest scop. Astfel, vom avea un algoritm care realizează o selectare *pe loc*. Această soluţie va fi avantajoasă doar dacă "evitarea" prelucrării acestor elemente, având valoarea aleasă va fi mai simplu de realizat decât verificarea proprietăţii P.

```
subalgoritm Selectare_4(n,X):
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      X[i] ← valoare
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Să considerăm câteva enunţuri de probleme:

P18. Să se realizeze un tabel cu fetele înscrise la olimpiada de informatică!

P19. Dat fiind un număr natural, să se genereze şirul tuturor divizorilor săi!

P20. Fie şirul mediilor semestriale ale elevilor unei clase în care s-a comis o eroare, în sensul că s-au calculat şi mediile copiilor care au rămas corigenţi. Ar trebui modificat şirul, astfel încât media elevilor corigenţi să fie înlocuită cu 0!

Rezolvarea problemei **P18** se poate realiza aplicând algoritmul `Selectare_1(n,X,buc,Y)`, presupunând că tipul `sir` desemnează un tablou unidimensional de elemente de tip **record**, unde printre câmpuri există şi câmpul sex, care în cazul fetelor are valoarea `'F'`.

```
procedure Tabel(n:Byte; elevi:sir;
          var nr:Byte; var fete:sir);
var i:Byte;
begin
  nr := 0;
  for i := 1 to n do
    if elevi[i].sex = 'F' then begin
      Inc(nr);
      fete[nr] := elevi[i]
    end
end;
```

Rezolvarea problemei P20 aparent este mai complicată, dar numai din cauza că proprietatea "corigent" nu este memorată explicit printre câmpurile articolelor, ci trebuie s-o stabilim pe baza mediilor pe discipline. Presupunem că programul conţine următoarele declaraţii:

```
const nrdiscipline = 15;
type unelev = record
       nume:string[30];
       medii:array[1..15] of 3..10;
       media:Real      {media generală}
     end;
type catalog = array[1..30] of unelev;
procedure Sterge(n:Byte; var c:catalog);
var i,j:Byte;,
begin
 for i := 1 to n do begin
  j:=1;
  while (j <= nrdiscipline) and
                (c[i].medii[j] >= 5) do Inc(j);
  if j <= nrdiscipline then
    c[i].media := 0
  end
end;
```

Probleme de LOGICĂ

Mihai Scorțaru

Nu credem că problemele de logică din numărul anterior al revistei v-au ridicat probleme deosebite. Totuși, dacă vreuna dintre soluții "v-a scăpat", veți găsi prezentarea ei în cadrul acestui articol.

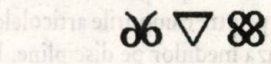
Șir "nerecurent"

Problema pare destul de ciudată la prima vedere. Nu pare a exista nici o legătură între simboluri; de fapt, nu există nici una. Totuși, se observă că toate cele cinci simboluri sunt simetrice față de o axă verticală care trece prin centrul lor. Dacă "tăiem" simbolurile în două și luăm în considerare doar partea dreaptă obținem următoarele cinci "simboluri":

1 2 3 4 5

Așadar, fiecare simbol este obținut dintr-un număr prin adăugarea în fața sa a imaginii oglindite a numărului respectiv.

În concluzie, următoarele trei simboluri corespund numerele 6, 7 și 8, deci următoarele trei elemente ale șirului sunt:



Iepurașul

Această problemă face parte dintr-o categorie clasică. Va trebui să alegem întrebarea astfel încât răspunsul să fie același indiferent dacă iepurașul va minți sau va spune adevărul și pe baza răspunsului să putem determina care drum duce în *Țara celor care mint tot timpul* și care duce în *Țara celor care spun adevărul întotdeauna.*

În această situație, o întrebare potrivită este: *Iepuraș drăgălaș, care este drumul pe care te vei întoarce în țara ta.*

Dacă *Iepurașul* spune adevărul, atunci va spune că se va întoarce în țara sa (*Țara celor care spun adevărul întotdeauna*) exact pe drumul care duce în acea țară.

Dacă *Iepurașul* minte, atunci va spune că se va întoarce în țara sa (*Țara celor care mint tot timpul*) pe drumul care duce în cealaltă țară.

Așadar, în ambele situații, drumul indicat de *Iepuraș* este cel care duce în *Țara celor care spun adevărul întotdeauna.*

Incrementare în C++

Cinci variante diferite de a realiza incrementări în C++ sunt prezentate în continuare:

- `i++;`
- `++i;`
- `i=i+1;`
- `i+=1;`
- `i-=-1;`

Dublarea valorii în C++

Această problemă a fost inspirată din ultima variantă de incrementare de la problema anterioară. Dacă numărul `1` este înlocuit cu `i`, atunci ceea ce se obține seamănă cu o "navă spațială". Așadar, instrucțiunea căutată este:

```
i-=-i
```

În urma executării acestei instrucțiuni valoarea variabilei scade cu `-i`, ceea ce este echivalent cu adunarea valorii `i`, deci rezultatul final va fi `2 * i`.

Întrebarea Microsoft

Evident, este foarte puțin probabil ca un programator care merge la un interviu de angajare să cunoască răspunsul la o astfel de întrebare.

Scopul acestei întrebări era verificarea faptului că intervievații pot să se descurce în situații dificile neașteptate. Cei mai mulți au prezentat modalitatea în care poate fi aflat numărul de benzinării apelând la diferite informații furnizate de diferite instituții ale statului.

Un programator a dat un răspuns care a fost mai apreciat decât cele "corecte", datorită "prezenței de spirit". El a spus, simplu, *Too many!*

Căsuța liberă

Legătura dintre cele trei numere dintr-un grup este următoarea: *numărul de jos este factorialul diferenței celor două numere de sus.* Așadar, răspunsul este `(7 - 2)! = 120`.

Nu pierde

ghidul tău în lumea tehnologiilor!

Publicarea acestei reviste este posibilă datorită sponsorizărilor acordate de următoarele firme care s-au decis să sprijine învățământul românesc de informatică

SPONSORI DE PLATINĂ

COMPAQ

SPONSORI DE AUR

FORTE COMPUTERS
Flexibility. Made in Romania

ASTRAL TELECOM

SPONSORI DE ARGINT

s&t
IT SOLUTIONS & SERVICES

hp
invent

ERICSSON

Magua

M.G.T.
EDUCATIONAL

a&C INTERNATIONAL S.A.
the CAD-CAM factory
autodesk
authorized distributor

ALCATEL

*Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

Martie 2003
Vol. 13/3

Lei 30.000

gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

serial

COMPRESIA datelor

Plus

internet
pagini web cu php4

actual
procesoarele viitorului

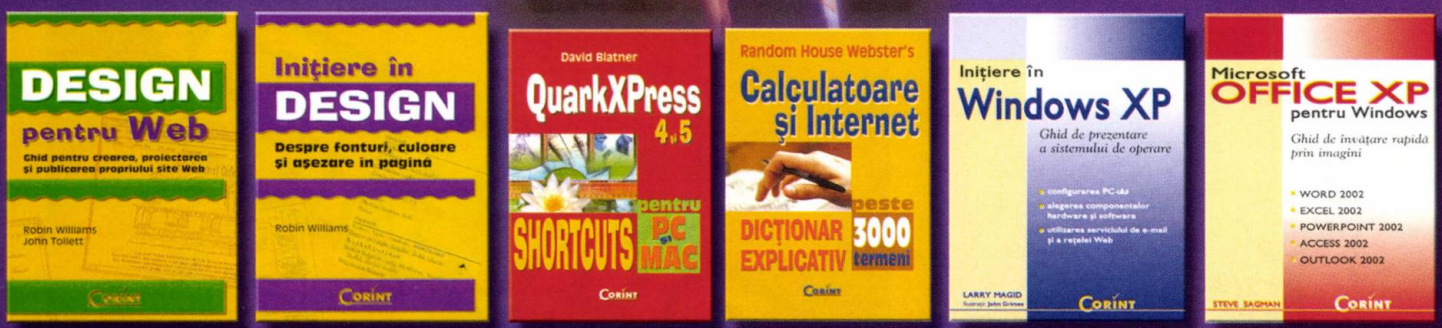
begin
teoreme de programare

Editura CORINT

El a aflat...
Voi nu vreţi?

COLECŢIA
IT

Lucrări sub licenţa prestigioaselor edituri:
Peachpit Press – Pearson Technology Group,
Random House Webster's



CORINT Difuzare: Str. Teodosie Rudeanu nr. 21,
Tel: 222 19 49; 223 19 28; Fax: 222 40 34;
E-mail: corint@dnt.ro

www.agora.ro
www.pcmagazine.ro
www.netreport.ro
www.eweek.ro
www.pcconcrete.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Vol. 13, Nr. 3 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Redactor-șef:
Mihai Scorțaru
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Claudiu Soroiu
Au mai colaborat:
Mihai Stroe (București)
conf. dr. *Marin Vlada (București)*
Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/192.422, 0265/166.516
Fax: 0265/166.290
http://www.ginfo.ro
ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 0265/162.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*", "*NetReport*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
Florin Oprescu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

actual.. PAG. 5

PROCESOARELE viitorului
Mihai SCORȚARU

probleme.. PAG. 7

Concursul INTER-JUDEȚEAN de programare INFO-OLTENIA
~ Drobeta Turnu-Severin ~

soluții.. PAG. 12

Faza LOCALĂ a olimpiadei de INFORMATICĂ
~ soluțiile problemelor ~
~ propuse spre rezolvare ~
~ în județul Cluj ~

concurs.. PAG. 18

Enunțurile problemelor de la rundele #13-#16

focus.. PAG. 22

PREPROCESAREA
Mihai STROE

babel.. PAG. 25

Conceptul de ALGORITM - abordare MODERNĂ
Marin VLADA

internet.. PAG. 32

Pagini WEB cu PHP 4
Mihai SCORȚARU
Claudiu SOROIU
~ funcții, clase, obiecte ~

profil.. PAG. 40

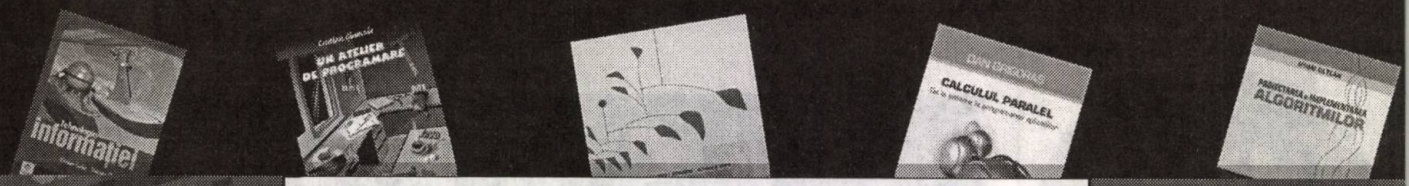
TUX din NORD
Carmen ȘUȘMAN
~ o scurtă istorie a ~
~ sistemului de operare ~
~ Linux ~
Despre pinguin

serial.. PAG. 42

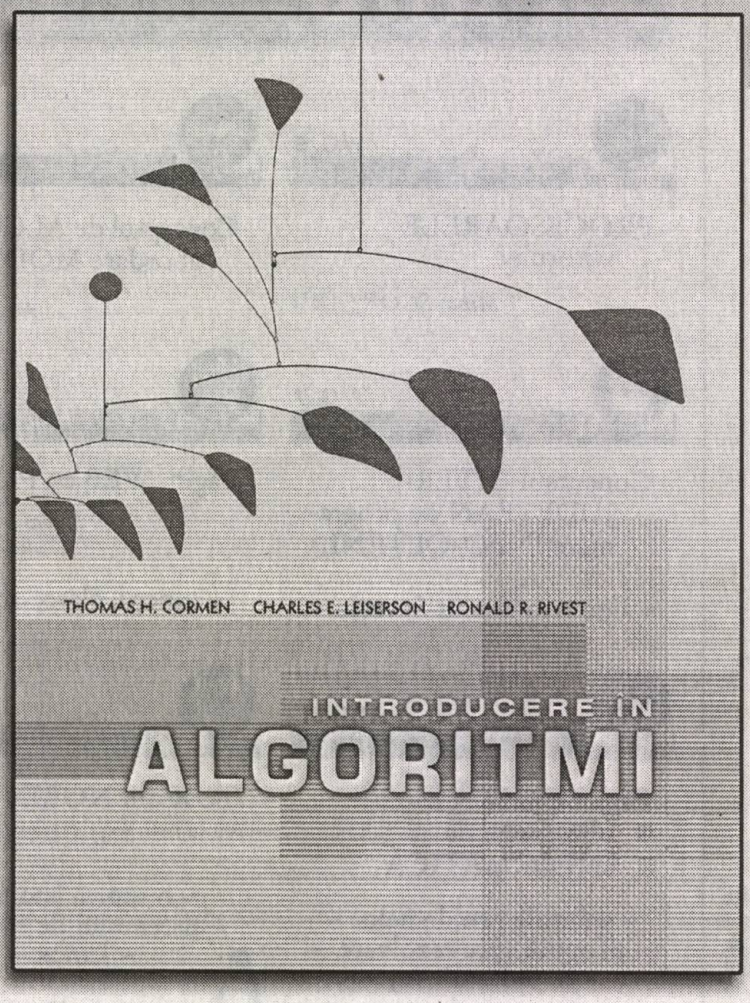
COMPRESIA datelor
Claudiu SOROIU

begin.. PAG. 47

TEOREME de programare
Clara IONESCU



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Steagul NEGRU

Existau acum câteva secole unele ambarcațiuni care arborau cu mândrie un steag negru pe care era desenat un craniu și două oase. Membrii echipajelor acestor nave erau legendarii pirați, cei a căror unic scop era jefuirea altor nave care "se aflau prin zonă". Prada era apoi fie ascunsă, fie comercializată în diverse porturi. Piraților aveau destui clienți deoarece mărfurile lor erau de o excelentă calitate și mult mai ieftine.

Acum piraților mărilor au dispărut, societatea umană a evoluat și, în ultimele decenii au apărut un alt tip de pirați. Meseria lor e mult mai simplă, cu mult mai puține riscuri, iar numărul clienților mult mai mare. Nu se mai fură mărfuri, ci informații stocate pe dispozitive electronice. Aurul, pietrele prețioase și delicatesele au fost înlocuite de aplicații, muzică, filme, jocuri etc.

Înainte piraților erau considerați oameni de temut, de care e bine să te ferești. Acum mulți dintre clienții lor îi consideră "oameni de bine". Mai mult, acum chiar și unii producători îi tolerează, ba chiar îi folosesc pentru a-și populariza produsele.

Într-una din zilele trecute am primit la redacție un *mail* prin care o "companie" încerca să-și facă reclamă. Mesajul conținea o adresă *Web* la care putea fi studiată oferta și completat un formular de comandă. În mesaj ni se promiteau ultimele produse *software* apărute și nu numai, la prețuri mici.

Pentru a vedea ce *soft*-uri se mai vând în *România* și la ce prețuri am vizitat *site*-ul respectiv. Nici urmă de aplicații *software*! În schimb, o mulțime de filme, albume audio și jocuri la prețuri promoționale.

Un album cu *Asia*, *Cargo* etc. costa sub 100.000 lei. Cele mai noi CD-uri aveau prețuri puțin mai mari, dar acceptabile. Un film costa 60.000 lei dacă ocupa un singur *CD*; dacă nu, pentru fiecare *CD* în plus se adăugau încă 60.000 lei.

Jocurile erau ceva mai scumpe: 80.000 lei pentru fiecare *CD*. Totuși, mult mai ieftin decât în alte părți unde, de regulă, prețul unui joc nou depășește suma de un milion de lei.

Pe *site* am găsit informații detaliate despre expedierea produselor (valoarea minimă a unei comenzi era de 300.000 lei), ni se promitea păstrarea confidențialității asupra comenzii etc. În plus, dacă nu eram mulțumiți de produse le puteam returna în termen de 30 de zile și primeam banii înapoi! Oare?

Site-ul nu era foarte bine realizat, formularul de comandă lăsa de dorit, dar totul era funcțional.

Așa că am zis să încerc și eu și am ales câteva "produse", dar cu atenție. Erau jocuri pentru care am deja licență (doar m-am lăudat într-un editorial anterior că dețin legal o grămadă de jocuri).

În formularul de comandă i-am informat pe acești pirați care vor să pară că lucrează legal că au încercat să-și facă reclamă chiar pe adresa unei reviste *IT*.

Am fost surprins de promptitudinea cu care mi s-a răspuns: "*Vă mulțumim pentru avertisment. Vă rugăm să ne comunicați adresa de e-mail la care ați primit mesajul cu reclama pentru a o șterge din baza de date! Comanda dumneavoastră va fi onorată în următoarele zile!*". Destul de interesant, nu? Adică, "m-ați prins, dar dacă tot m-ați prins nu vreți să cumpărați puțină marfă?".

Nu am mai răspuns nimic fiindcă nu mai aveam nimic de zis. După câteva ore am primit un nou mesaj în care era precizată data la care va fi expediat coletul cu produsele comandate de mine. Acum sunt curios! Din păcate coletul va ajunge doar după închiderea ediției pentru acest număr, așa că nu știu încă ce va conține coletul.

Așadar, cam așa se vinde *soft* în *România*. Oricine poate accesa *site*-ul respectiv și poate obține tot ce vrea. Din motive lesne de înțeles, nu voi publica adresa *site*-ului prin intermediul căruia această "companie" își comercializează "produsele".

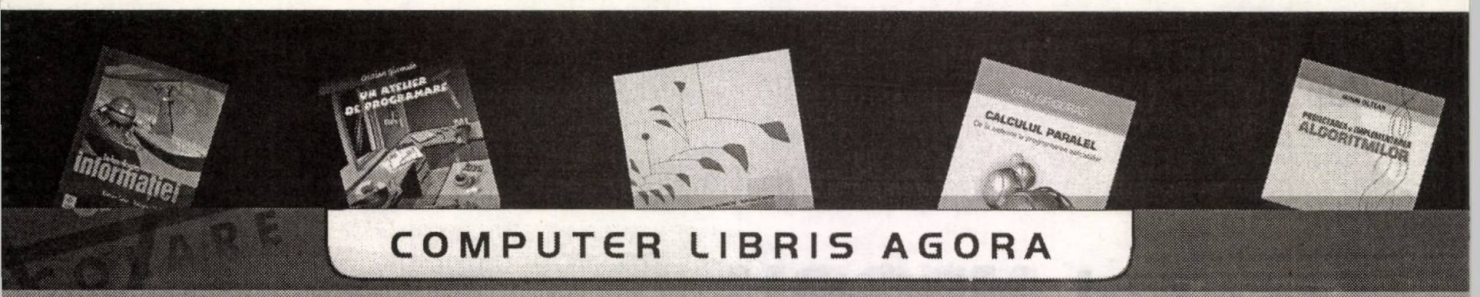
Trebuie remarcat faptul că această "companie" dorește să pară că funcționează perfect legal. Mesajul inițial pe care l-am primit conținea un text în care se spunea că primesc *mail*-ul respectiv deoarece l-am solicitat (eu nu îmi aduceam aminte, dar poate am uitat!) și acum am devenit abonat. În plus, mi se oferea posibilitatea de a accesa un *link* prin intermediul căruia puteam să renunț la abonament.

Din nefericire, mulți necunoscători chiar cred că "produsele" obținute pe astfel de căi sunt perfect legale. În plus, nu se face mai nimic pentru a împiedica astfel de practici.

Nu pot să cred că nici unul dintre cei care ar trebui să acționeze împotriva pirateriei nu a căutat pe *Internet site*-uri cu astfel de oferte promoționale. Nu pot să îmi imaginez cum poate funcționa o astfel de "companie". Dar, suntem în *România*...

Așadar, drapelul negru flutură în continuare chiar sub nasul nostru. Singura diferență este faptul că acum e doar un steag virtual...

Mihai Scorțaru



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

PROCESOARELE viitorului

Mihai Scorţaru

Recent, au apărut mai multe informaţii referitoare la planurile de viitor ale companiilor Intel şi AMD. Evident, nici unul dintre cei doi participanţi la războiul procesoarelor nu a confirmat sau infirmat aceste zvonuri. În cadrul acestui artciol vă vom prezenta câteva detalii cunoscute în acest moment despre procesoarele care vor apărea în următorii ani.

Athlon XP 3000+

Vom începe cu prezentarea unui procesor care a fost lansat deja la începutul lunii februarie. Este vorba de un nou tip de procesor *Athlon XP*, bazat pe un nou nucleu, cunoscut până acum sub numele de cod *Barton*.

Principala îmbunătăţire adusă procesoarelor *Athlon XP* este dublarea memoriei *cache level* 2 de la 256 KB la 512 KB. Celelalte carcateristici sunt: magistrala de 333 MHz, dimensiunea de 101 mm^2, 54,3 milioane de tranzistori şi tensiunea de funcţionare de 1,65 V.

Au fost lansate pe piaţă trei modele, având indicativii de performanţă 2500+, 2800+, respectiv 3000+. Frecvenţele reale de funcţionare ale celor trei modele sunt 1,8 GHz, 2,083 GHz şi 2,167 GHz.

Trebuie remarcat faptul că, procesoarele *Athlon XP* anterioare (cunoscute sub numele de cod *Thoroughbred*), care funcţionau la aceste frecvenţe, aveau indicativii de performanţă 2200+, 2600+, respectiv 2700+.

Aşadar, *AMD* consideră că dublarea memoriei *cache* duce la o creştere importantă a performanţelor procesoarelor. Din nefericire, această afirmaţie a fost infirmată de mai multe teste efectuate, noul *Athlon XP* 3000+, bazat pe nucleul *Barton*, fiind mai puţin performant decât vechiul *Athlon XP* 2800+, bazat pe nucleu *Thoroughbred*. Aceasta se datorează, în mare parte, faptului că frecvenţa de funcţionare a vechiului *Athlon XP* 2800+ este mai mare decât cea a noului *Athlon XP* 3000+ (2,25 GHz faţă de 2,167 GHz).

Indicativul 3000+ sugerează faptul că performanţele sunt comparabile cu cele ale unui procesor *Pentium* 4 cu frecvenţa de 3,06 GHz. Din nefericire, există foarte puţine cazuri în care un *Athlon XP* 3000+ poate ţine pasul cu cel mai performant procesor de la *Intel*.

În concluzie, *AMD* a exagerat puţin valoarea indicativului de performanţă al noului procesor, ceea ce a dus la apariţia multor critici. Această tactică este datorată faptului că *AMD* nu mai reuşeşte să ofere produse de nivelul celor fabricate de *Intel*.

Următorul procesor

Următoarea modificare pe care o va efectua *AMD* pentru procesoarele sale va fi, probabil, trecerea de la magistrala de 333 MHz la cea de 400

Procesoarele Intel

Procesor	Frecvenţă	Magistrală	Preţ SUA	Preţ România
Pentium 4	1,6 GHz	400 MHz	106$	4,9 mil. lei
Pentium 4	1,7 GHz	400 MHz	114$	5,0 mil. lei
Pentium 4	1,8 GHz	400 MHz	103$	5,4 mil. lei
Pentium 4	1,9 GHz	400 MHz	140$	6,6 mil. lei
Pentium 4	2,0 GHz	400 MHz	145$	6,4 mil. lei
Pentium 4	2,2 GHz	400 MHz	163$	8,6 mil. lei
Pentium 4	2,26 GHz	533 MHz	170$	8,8 mil. lei
Pentium 4	2,4 GHz	400 MHz	145$	8,1 mil. lei
Pentium 4	2,4 GHz	533 MHz	156$	7,5 mil. lei
Pentium 4	2,66 GHz	533 MHz	236$	11,3 mil. lei
Pentium 4	2,8 GHz	533 MHz	353$	15,6 mil. lei
Pentium 4	3,06 GHz	533 MHz	564$	26,0 mil. lei
Celeron	1,0 GHz	100 MHz	40$	1,7 mil. lei
Celeron	1,1 GHz	100 MHz	39$	1,7 mil. lei
Celeron	1,2 GHz	100 MHz	39$	1,8 mil. lei
Celeron	1,3 GHz	100 MHz	46$	1,8 mil. lei
Celeron	1,7 GHz	400 MHz	54$	2,2 mil. lei
Celeron	1,8 GHz	400 MHz	66$	2,8 mil. lei
Celeron	2,0 GHz	400 MHz	82$	3,2 mil. lei
Celeron	2,1 GHz	400 MHz	90$	4,2 mil. lei

MHz. Un procesor care să folosească o astfel de magistrală va fi lansat cel mai devreme în luna mai. Indicativul de performanță ar putea fi 3200+, dar frecvența reală de funcționare nu este cunoscută încă.

Intel nu va mări frecvența de funcționare a celui mai rapid procesor al său decât în mai sau iunie, când va apărea un *Pentium* 4 bazat pe actualul nucleu *Northwood* care va funcționa la frecvența de 3,2 GHz.

Următoarea noutate însă va apărea în cursul lunii aprilie când ar putea fi lansate modele *Pentium* 4 care folosesc o magistrală de 800 MHz, dar au frecvențe de funcționare de numai 3 GHz. În luna mai vor apărea și alte procesoare care folosesc o astfel de magistrală, dar frecvențele vor fi mai mici (între 2,4 GHz și 2,8 GHz).

Generația următoare

Numele viitoarelor procesoare *AMD* care vor aduce modificări semnificative este deja cunoscut: *Athlon* 64.

După cum probabil știți, *Athlon* 64 va folosi o arhitectură nouă pe 64 de biți și va fi lansat în toamna acestui an (cel mai probabil în septembrie).

Aceste procesoare vor avea, în momentul lansării, indicativul de performanță 3400+ și frecvența reală de doar 2 GHz. Vor fi lansate, succesiv, procesoare cu indicativii de performanță de până la 4400+, dar frecvența reală nu va depăși 3 GHz.

Numele de cod al următoarei familii de procesoare *Intel* este *Prescott*. Numele comercial nu este cunoscut încă, dar s-ar putea ca acesta să fie tot *Pentium* 4.

Procesoarele *Prescott* vor fi fabricate pe baza unei tehnologii de 0,09 µm, vor avea magistrala de 800 MHz, o memorie cache de 1 MB și suport pentru tehnologia *HyperThreading* II. Aceste procesoare vor apărea la sfârșitul acestui an la o frecvență inițială de 3,2 GHz. În total vor fi zece modele, frecvența maximă care va fi atinsă fiind de 5 GHz.

Va exista și o variantă *B* a procesoarelor *Prescott* care vor folosi o magistrală de 1,06 GHz. *Prescott B* va apărea doar în 2005 și vor exista doar trei modele care vor funcționa la frecvențe de 4,8 GHz, 5,06 GHz, respectiv 5,33 GHz.

Viitorul mai îndepărtat

În anul 2004 *AMD* va lansa o nouă versiune a procesoarelor *Athlon* 64, cunoscută acum sub numele de cod *San Diego*.

Aceste procesoare vor fi fabricate pe baza unei tehnologii de 0,09 µm și vor avea, pentru început, indicativul de performanță 4000+, frecvența reală fiind de 2,6 GHz.

Vor exista alte patru modele, cel mai rapid dintre acestea funcționând la frecvența de 3,4 GHz și având indicativul de performanță 4800+.

Despre viitoarele procesoare *Intel* sunt disponibile mai multe informații. Succesorul procesorului *Prescott* va apărea, probabil, în 2005 și este cunoscut sub numele de cod *Tejas*.

Există cel puțin un aspect interesant referitor la acest procesor, și anume faptul că nu va avea pini. *Tejas* va utiliza un nou tip de *socket*, numit *LGA* (*Land Grid Array*). Noul *socket* va fi format din 775 de puncte de contact fără pini (engl. *pinless contacts*).

Contactul și fixarea procesorului va fi realizată prin intermediul unui *socket body stiffener*. Această componentă va preveni montarea incorectă a procesorului.

Tejas va avea o magistrală de 1,06 GHz, va fi fabricat pe baza tehnologiei de 0,09 µm, va avea o memorie *cache* de 2 MB și va debuta la frecvența de 5,33 GHz.

Vor exista alte trei modele care vor funcționa la frecvențe de 5,6 GHz, 5,86 GHz, respectiv 6,13 GHz.

La sfârșitul anului 2005 va apărea o nouă versiune a procesorului *Tejas* care va utiliza o magistrală de 1,2 GHz și va fi fabricat pe baza unei tehnologii de 0,065 µm.

Frecvența inițială a acestui procesor va fi de 6 GHz, dar vor exista modele care vor ajunge la 7,2 GHz.

Zvonuri

Evident, toate procesoarele prezentate există la ora actuală doar pe hârtie și majoritatea informațiilor prezentate nu sunt confirmate.

Există o destul de mare probabilitate ca planurile de viitor ale celor două companii să se schimbe.

Concluzii

Lupta pentru supremație în industria procesoarelor pare a nu mai fi echilibrată. Se pare că *Intel* are un avantaj important și este destul de dificil pentru *AMD* să recupereze.

Există prognoze potrivit cărora în cel mult cinci ani *AMD* va pierde definitiv bătălia și va trebui să se retragă. Se pare că *IBM* dorește să intre pe piața procesoarelor, așa că achiziționarea companiei *AMD* de către gigantul *IBM* nu este exclusă.

Dificultățile cu care se confruntă *AMD* sunt ilustrate de prețurile procesoarelor sale și de imposibilitatea satisfacerii tuturor cererilor. În casetele de pe aceste pagini sunt prezentate modelele de procesoare disponibile, oferite de *Intel* și *AMD*, precum și prețurile orientative în *România* și *S.U.A.* Prețurile erau valabile în prima săptămână a lunii martie.

Procesoarele Athlon XP

Cod	Indicativ	Frecvență	Preț SUA	Preț România
Palomino	1600+	1,4 GHz	49$	2,2 mil. lei
Palomino	1700+	1,46 GHz	48$	2,2 mil. lei
Palomino	1800+	1,53 GHz	55$	2,6 mil. lei
Palomino	1900+	1,6 GHz	63$	3,1 mil. lei
Palomino	2000+	1,66 GHz	69$	3,2 mil. lei
Palomino	2100+	1,73 GHz	78$	3,6 mil. lei
Thoroughbread	2200+	1.8 GHz	95$	4,4 mil. lei
Thoroughbread	2400+	2,0 GHz	134$	6,8 mil. lei
Thoroughbread	2600+	2,08/2,13 GHz	226$	10,5 mil. lei
Thoroughbread	2700+	2,17 GHz	257$	12,4 mil. lei
Thoroughbread	2800+	2,25 GHz	367$	21,5 mil. lei
Barton	2500+	1,8 GHz	179$	n/a
Barton	2800+	2,08 GHz	389$	17,3 mil. lei
Barton	3000+	2,16 GHz	571$	n/a

Concursul INTERJUDEȚEAN de programare INFO-OLTENIA

Ediția din acest an a concursului Info-Oltenia s-a desfășurat în perioada 7-8 februarie la Colegiul Național "Traian" din Drobeta Turnu-Severin. Vă prezentăm în continuare enunțurile problemelor de la concursul de programare individual și de la cel pe echipe.

P030301: Cuburi perfecte

clasa a IX-a, concursul individual
prof. Doru Popescu Anastasiu

Se consideră două numere naturale n și k. Să se determine o modalitate de a scrie valoarea n ca sumă de k numere naturale nenule astfel încât să fie respectate următoarele proprietăți:

- dintre aceste numere cel mult unul dintre ele nu este cub perfect;
- cel mult două dintre aceste numere sunt egale.

Un număr natural m este *cub perfect* dacă există un număr natural k astfel încât $m = k^3$.

Date de intrare

Valorile n și k vor fi citite de la intrarea standard.

Date de ieșire

La ieșirea standard se va afișa, pe o linie, unul din mesajele DA (dacă există cel puțin o soluție) sau NU (dacă nu există nici o soluție). În cazul în care există soluții pe rândul următor se va afișa șirul de numere (o singură soluție) în ordine crescătoare.

Restricții

- $1 \le n \le 2000000000$;
- $1 \le k \le 20000$.

Exemplu

Intrare

```
96
3
```

Ieșire

```
DA
5 27 64
```

Timp de execuție: 1 secundă/test

P030302: Sărituri

clasa a IX-a, concursul individual
prof. Laviniu Bejenaru

La un concurs pentru androizi, fiecare este suspus următorului test: pe o suprafață dreptunghiulară cu m linii și n coloane conținând $m \times n$ zone de energie, un concurent poate sări dintr-o zonă de energie în orice altă zonă culegând energia existentă din zona în care sare.

Orice salt poate fi efectuat numai dintr-o zonă cu energie mai mică într-o zonă cu energie mai mare. La fiecare salt are loc un consum energetic egal cu diferența dintre valoarea energetică a zonei în care se sare și valoarea energetică a zonei din care se sare.

Desigur, pentru fiecare salt, concurentul trebuie să aibă în momentul saltului o rezervă de energie mai mare sau egală cu energia necesară saltului.

Concurentul se găsește inițial în zona aflată pe linia 1 și coloana 1 de energie nenulă, deținând energia existentă în acea zonă.

Scopul testului este acela de a atinge una din zonele cu energia cea mai mare de pe întreaga suprafață efectuând un număr minim de sărituri.

Date de intrare

Prima linie a fișierului de intrare **INPUT.TXT** conține valorile m și n care reprezintă numărul de linii, respectiv nu-

mărul de coloane ale suprafeței; cele două valori vor fi separate printr-un singur spațiu.

Fiecare dintre următoarele m linii va conține câte n numere întregi, separate prin spații. Aceste numere reprezintă valorile energetice corespunzătoare zonelor suprafeței.

Date de ieșire

În cazul în care există posibilitatea de a se ajunge într-o zonă cu valoarea energetică maximă, fișierul **OUTPUT.TXT** va conține o singură linie pe care se va afla numărul minim p al pașilor (salturilor) necesari pentru a ajunge în zona respectivă. Dacă nu se poate atinge o astfel de zonă, fișierul de ieșire va conține doar valoarea -1.

Restricții și precizări

- $2 \le m, n \le 50$;
- valoarea energetică a fiecărei zone este un număr natural cuprins între 0 și 10000.

Exemplu

INPUT.TXT

```
3 3
1 8 2
4 5 0
10 9 2
```

OUTPUT.TXT

```
4
```

Timp de execuție: 1 secundă/test

P030303: Punct final

clasa a X-a, concursul individual
prof. Roxana Tîmplaru

Verde Împărat vrea să-și mărite fata cu cel mai isteț pretendent. Pentru aceasta, cavalerii trebuie să rezolve următoarea problemă:

Dându-se un labirint cu $m \times n$ celule, un punct de plecare și cunoscându-se de câte ori trebuie să se treacă prin fiecare celulă pentru a se ajunge la locul în care se află fiica împăratului, se cere să se identifice poziția acesteia.

Se știe că nu sunt celule interzise, iar dintr-o celulă se poate ajunge doar într-o celulă vecină cu aceasta, urmând una din direcțiile: nord, sud, est sau vest.

Numărul de treceri prin fiecare celulă trebuie să fie cel specificat.

Pentru a se edifica, împăratul cere fiecărui cavaler să precizeze o singură poziție în care se poate afla fiica sa, chiar și pentru cazul în care sunt mai multe soluții. Întotdeauna există cel puțin o soluție.

Ajutați-l pe împărat să se hotărască!

Date de intrare

Prima linie a fișierului de intrare **PUNCT.IN** conține valorile m și n care reprezintă numărul de linii, respectiv numărul de coloane ale labirintului; cele două valori vor fi separate printr-un singur spațiu.

Fiecare dintre următoarele m linii va conține câte n numere întregi, separate prin spații. Aceste numere reprezintă numărul de treceri prin celulele corespunzătoare.

Date de ieșire

Fișierul de ieșire **PUNCT.OUT** va conține o singură linie pe care se vor afla două numere întregi, separate printr-un spațiu, care reprezintă linia și coloana punctului în care se află fiica împăratului.

Restricții și precizări

- $1 \le m, n \le 10$;
- fiecare element din matrice are cel mult valoarea 3, suma elementelor fiind mai mică sau egală cu 30.

Exemplu

PUNCT.IN

```
3 4
2 2 0 0
1 2 0 0
0 1 0 0
1 1
```

PUNCT.OUT

```
2 1
```

Timp de execuție: 10 secunde/test

P030304: Codificare

clasa a X-a, concursul individual
prof. Gabriela Zagara

Doi prieteni din orașe diferite corespondează între ei. Pentru ca mesajele lor să nu fie înțelese de ceilalți se hotărăsc să le codifice astfel: fiecare mesaj va conține două șiruri de numere (numerele având cel mult două cifre) și un șir de cuvinte.

Cel care primește mesajul va selecta toate numerele din primul șir care se regăsesc în cel de-al doilea, în aceeași ordine, astfel încât numărul lor să fie maxim. Decodificarea va fi apoi simplu de făcut. Din șirul de cuvinte se vor alege doar acele cuvinte, ale căror poziții pot fi identificate printre numerele selectate. Cuvintele vor fi afișate în ordinea în care apar în șirul de cuvinte inițial.

Date de intrare

Fișierul de intrare **COD.IN** are următoarea structură:

- pe prima linie se află valoarea n_1 (dimensiunea primului șir de numere) ;
- pe a doua linie se află termenii primului șir, separați prin câte un spațiu;
- pe a treia linie se află n_2 (dimensiunea celui de-al doilea șir de numere);
- pe a patra linie se află termenii celui de-al doilea șir, separați prin câte un spațiu;

- pe linia următoare se află m (numărul de cuvinte);
- pe următoarele m linii se află cele m cuvinte (câte un cuvânt pe linie).

Date de ieșire

Fișierul de ieșire **COD.OUT** va conține cuvintele decodificate, câte unul pe linie.

Restricții și precizări

- $1 < n_1, n_2 < 100$;
- $1 < m < 100$;
- dacă există mai multe soluții, se va genera doar una dintre ele;
- un cuvânt are cel mult 200 litere.

Exemplu

```
COD.IN
4
1 4 7 2
4
1 7 2 4
10
maine
stai
apa
soare
bani
are
acasa
ploaie
teza
casa

COD.OUT
maine
stai
acasa
```

Timp de execuție: 1 secundă/test

P030305: Caramele

clasele a XI-a și a XII-a, concursul individual
prof. Rodica Pintea

Cu ocazia împlinirii venerabilei vârste de 33 de ani, *Lolek* și *Bolek*, îndrăgitele personaje ale desenului animat, au primit din partea admiratorilor cadouri surpriză. Dar fiindcă le-a mers vestea că devorează mari cantități de caramele, cadourile nu au reprezentat nici o surpriză: fiecare cadou consta dintr-o punguță cu caramele *Karamka*.

Problema care le umbrește însă aniversarea este următoarea: *cum să împartă caramelele*? Fiecare vrea tot atâtea punguțe ca și celălalt și tot atâtea caramele ca și celălalt. Cum prietenii lor mai pricepuți le-au demonstrat că, fără a muta caramele dintr-o punguță în alta, acest lucru nu este totdeauna posibil, *Bolek* a acceptat în ultimă instanță să primească mai puține caramele decât *Lolek*, dar cât mai multe cu putință, însă în nici un caz să nu primească mai puține punguțe decât acesta (lucru posibil deoarece numărul total de punguțe primite este par) și nici să nu umble la caramelele din punguțele lui.

Date de intrare

Prima linie a fișierului de intrare **CARAMELE.IN** va conține numărul natural n, reprezentând numărul de punguțe primite în total de cele două personaje.

Cea de-a doua linie a fișierului va conține n numere naturale $c_1, c_2, ..., c_n$ reprezentând numărul de caramele din cele n punguțe. Aceste numere vor fi separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **CARAMELE.OUT** va conține o singură linie pe care se vor afla numerele de ordine ale punguțelor de caramele primite de *Lolek* (sau de *Bolek*, dacă îl simpatizați pe el mai mult), astfel încât diferența dintre numărul de caramele primite de *Lolek* și numărul de caramele primite de *Bolek* să fie minimă.

Restricții și precizări

- $2 \le n \le 40$, n este număr par;
- o punguță poate conține cel mult 40 de caramele;
- dacă există mai multe soluții, trebuie generată doar una dintre ele.

Exemplu

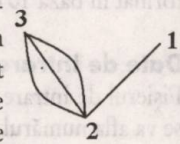
```
CARAMELE.IN
6
5 8 2 11 4 3

CARAMELE.OUT
1 2 5
```

Timp de execuție: 2 secunde/test

P030306: Multigraf

clasele a XI-a și a XII-a, concursul individual
prof. Doru Popescu Anastasiu

Fie n un număr natural și $d_1, d_2, ..., d_n$, n numere naturale nenule. Se cere să se verifice dacă există un multigraf cu n noduri care să aibă ca grade ale nodurilor sale numerele $d_1, d_2, ..., d_n$ (nu neapărat în această ordine).



Singura deosebire dintre un graf neorientat și un multigraf este aceea că, în multigraf, între două noduri distincte pot exista zero, una sau mai multe muchii. De exemplu, pot exista trei muchii între nodurile 2 și 3 (vezi figura).

Date de intrare

Prima linie a fișierului de intrare **MGRAF.IN** va conține numărul n al nodurilor multigrafului.

Cea de-a doua linie a fișierului va conține numerele d_1, d_2, ..., d_n separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **MGRAF.OUT** va conține câte o linie pentru fiecare muchie a multigrafului. Pe fiecare linie se vor afla câte două numere din mulțimea {1, 2, ..., *n*} care reprezintă extremitățile unei muchii.

Dacă nu există nici o soluție, fișierul va conține o singură linie pe care se va afla valoarea 0.

Restricții și precizări

- $1 \le n \le 500$;
- dacă există mai multe soluții, trebuie generată doar una dintre ele;
- gradele nodurilor sunt numere întregi cuprinse între 1 și 100.

Exemple

MGRAF.IN

```
4
1 2 10 2
```

MGRAF.OUT

```
0
```

MGRAF.IN

```
4
1 3 3 3
```

MGRAF.OUT

```
1 2
4 1
4 3
4 3
1 3
```

Timp de execuție: 1 secundă/test

P030307: Baza

clasele a IX-a și a X-a, concursul pe echipe
prof. Mihaela Runceanu

Se dă un număr natural ale cărui cifre pot fi 0, ..., 9 și litere latine A, ..., Z cu semnificația A = 10, B = 11 etc.

Se cere să se determine cel mai mic număr natural *k*, astfel încât numărul dat, fiind scris în baza *k*, dacă este transformat în baza 10 este divizibil cu *k* - 1.

Date de intrare

Fișierul de intrare **DATE.IN** conține o singură linie pe care se va afla numărul.

Date de ieșire

Fișierul de ieșire **DATE.OUT** va conține o singură linie pe care se va afla numărul *k*. Dacă nu există nici o soluție, în fișier se va scrie valoarea 0.

Restricții și precizări

- $2 \le k \le 36$;
- dacă există mai multe soluții, trebuie generată doar una dintre ele;
- numărul poate conține cel mult 1000000 de cifre.

Exemplu

DATE.IN

```
A1A
```

DATE.OUT

```
22
```

Timp de execuție: 1 secundă/test

P030308: Patinoare

clasele a IX-a și a X-a, concursul pe echipe
prof. Roxana Tîmplaru

Primarul orașului *Drobeta Turnu-Severin* dorește să construiască cel puțin un patinoar de formă pătrată într-unul din cartierele de blocuri ale orașului care are o formă pătrată. Ajutați-l pe primar să amplaseze patinoarul de formă pătratică printre blocurile din oraș, astfel încât suprafața sa să fie maximă.

Primarul dorește însă să găsească două amplasamente posibile, dacă există, ambele având suprafața maximă, urmând apoi să decidă în ședința cu consilierii unde va amplasa patinoarul, sau poate că va reuși să le amplaseze pe amândouă.

Se consideră orașul reprezentat sub forma unei matrice pătratice, cu $n \times n$ celule, în care sunt marcate cu 1 celulele ocupate de blocuri, iar cu 0 celulele libere. În cazul în care există mai mult de două amplasamente posibile, se vor prezenta doar două soluții prin latură și colțul din stânga jos, ținând cont de faptul că cele două pătrate pot să aibă puncte comune interioare sau pe latură.

Se știe că poate fi amplasat întotdeauna un patinoar de latură cel puțin 1.

Ajutați-l pe primar să se hotărască!

Date de intrare

Prima linie a fișierului de intrare **PATINOAR.IN** conține numărul *n* al liniilor și coloanelor matricei.

Fiecare dintre următoarele *n* linii va conține câte *n* numere, reprezentând elementele matricei. O poziție ocupată de un bloc este reprezentată de valoarea 1, iar o poziție liberă este reprezentată de valoarea 0. Numerele de pe o linie sunt separate prin spații.

Date de ieșire

Prima linie a fișierului **PATINOAR.OUT** va conține lungimea laturii celui mai mare pătrat în care poate fi amplasat patinoarul.

Cea de-a doua linie va conține valoarea 1 dacă există o singură posibilitate de amplasare, sau 2 dacă există cel puțin două posibilități.

Dacă valoarea de pe a doua linie este 1, atunci linia următoare va conține linia și coloana colțului din stânga jos (valori separate printr-un singur spațiu) al poziției de amplasare a patinoarului.

Dacă valoarea de pe a doua linie este 2, atunci următoarele două linii vor conține coordonatele a două poziții în care poate fi amplasat patinoarul.

Restricție

- $1 \le n \le 100$.

Exemplu

PATINOAR.IN

```
5
1 0 1 0 0
0 0 0 1 0
0 0 0 1 0
1 0 0 1 0
0 1 1 0 0
```

PATINOAR.OUT

```
2
2
3 1
4 2
```

Timp de execuție: 1 secundă/test

P030309: Poligoane

clasele a XI-a și a XII-a, concursul pe echipe
prof. Doru Popescu Anastasiu

Se consideră două poligoane convexe descrise prin coordonatele vârfurilor lor (în sens corect de citire - sens trigonometric sau antitrigonometric). Se cere să se determine perimetrul intersecției celor două poligoane.

Date de intrare

Prima linie a fișierului de intrare **POLI.IN** conține numărul n_1 al vârfurilor primului poligon.

Următoarele n_1 linii conțin perechi de numere separate prin câte un spațiu care reprezintă coordonatele vârfurilor primului poligon.

Următoarea linie conține numărul n_2 al vârfurilor celui de-al doilea poligon.

Următoarele n_2 linii conțin perechi de numere separate prin câte un spațiu care reprezintă coordonatele vârfurilor celui de-al doilea poligon.

Date de ieșire

Fișierul **POLI.OUT** va conține o singură linie pe care se va afla un singur număr real (trunchiat la 3 zecimale) care reprezintă perimetrul cerut.

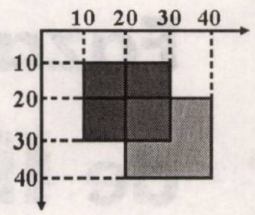
Restricții și precizări

- $1 \le n_1, n_2 \le 30$;
- coordonatele sunt date în ordinea: abscisă, ordonată;
- abscisele punctelor sunt numere naturale cuprinse între 0 și 640.
- ordonatele punctelor sunt numere naturale cuprinse între 0 și 480.

Exemplu

POLI.IN

```
4
10 10
30 10
30 30
10 30
4
20 20
40 20
40 40
20 40
```



POLI.OUT

```
40.000
```

Timp de execuție: 1 secundă/test

P030310: Numere importante

clasele a XI-a și a XII-a, concursul pe echipe
prof. Rodica Pintea

Numim *număr important* un număr natural k cu proprietatea că numărul de zerouri din reprezentarea sa binară este cel puțin egal cu numărul de cifre 1 din această reprezentare. De exemplu 10 (1010_2) este un număr important, dar 6 (110_2) nu este un număr important.

Pentru un număr natural n dat ($n \le 2000000000$) se cere să se afișeze câte numere importante mai mici sau egale cu n există.

Date de intrare

Fișierul de intrare **NRIMP.IN** va conține o singură linie pe care se află valoarea n.

Date de ieșire

La ieșirea standard se va afișa numărul valorilor mai mici sau egale cu n care reprezintă numere importante.

Exemplu

NRIMP.IN

```
12
```

Ecran

```
7
```

Explicație

Valorile $0 = 0_2$, $2 = 10_2$, $4 = 100_2$, $8 = 1000_2$, $9 = 1001_2$, $10 = 1010_2$, $12 = 1100_2$ sunt numerele importante mai mici sau egale cu 12.

Timp de execuție: 1 secundă/test

Cluj-Napoca, 1 februarie 2003

Faza LOCALĂ a olimpiadei de INFORMATICĂ

Vă prezentăm în continuare soluțiile celor nouă probleme propuse spre rezolvare la ediția 2003 a fazei locale din județul Cluj a olimpiadei de informatică.

P020307: Ecuație

Ecuația $x^2 - y^2 = k$ poate fi scrisă sub forma $(x + y) \cdot (x - y) = k$. Datorită faptului că x și y sunt valori întregi, $x + y$ și $x - y$ sunt, de asemenea, valori întregi.

Vom considera toate posibilitățile de a scrie valoarea k ca produs de numere întregi $k = p \cdot q$. Pentru fiecare posibilitate va trebui să rezolvăm sistemul de ecuații:

$$\begin{cases} x + y = p \\ x - y = q \end{cases}.$$

Soluțiile acestui sistem de ecuații sunt:

$$\begin{cases} x = \frac{p+q}{2} \\ y = \frac{p-q}{2} \end{cases}.$$

Datorită faptului că soluțiile trebuie să fie numere întregi, se observă că valorile p și q trebuie să aibă aceeași paritate.

Produsul a două numere de aceeași paritate are întotdeauna paritatea celor două numere; așadar, pentru identificarea numerelor p și q vom considera toate numerele mai mici decât k care au aceeași paritate cu k.

Vom considera toate variantele posibile pentru valoarea p și vom determina, pentru fiecare dintre acestea, valoarea q folosind formula $q = k / n$.

Se observă că dacă x și y sunt soluții ale ecuației, atunci perechile $(-x, y)$, $(x, -y)$ și $(-x, -y)$ sunt și ele soluții. Așadar, va trebui să determinăm doar soluțiile pozitive și în momentul identificării unei astfel de soluții să le generăm și pe celalte trei.

O situație specială este cazul în care una dintre valorile x și y este 0. Nu există posibilitatea ca ambele valori să fie 0, deoarece k este întotdeauna un număr nenul.

În această situație, doar două dintre cele patru perechi duc la obținerea unor soluții distincte. Din acest motiv vom genera doar perechea $(-x, -y)$ care este, cu siguranță, diferită de perechea (x, y) indiferent care dintre cele două valori este 0.

Se observă că este suficient să luăm în considerare doar divizorii p pentru care $p \le q = k / p$ deoarece perechea (p, q) duce la obținerea acelorași soluții ca și perechea (q, p). Așadar, vom lua în considerare doar divizorii lui k care sunt mai mici sau egali cu rădăcina pătrată a acestui număr.

Analiza complexității

Intrarea constă în citirea unui singur număr, așadar ordinul de complexitate al operației de citire a datelor este $O(1)$.

Datorită faptului că pentru fiecare divizor determinat efectuăm un număr constant de operații și vom verifica cel mult $\sqrt{k}$ / 2 divizori, ordinul de complexitate al operației de determinare a soluțiilor ecuației este $O(\sqrt{k})$.

Datele de ieșire sunt scrise pe parcursul determinării soluțiilor, deci operația de scriere a acestora nu consumă timp suplimentar.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(1) + O(\sqrt{k}) = O(\sqrt{k})$.

Dificultate 10

Algoritmi	★☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	★☆☆☆☆
Cunoștințe	★☆☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/ecuatie.cpp

P020308: Număr

Pentru a determina cel mai mic număr par care este format din anumite cifre date vom ordona crescător aceste cifre.

Pentru ca numărul obținut să nu înceapă cu zerouri nesemnificative vom muta cea mai mică cifră pe prima poziție.

Pentru ca numărul să fie par vom muta cea mai mare cifră pară pe ultima poziție cu excepția situației în care aceasta a fost mutată pe prima poziție. În cazul în care cifra respectivă a ajuns pe prima poziție știm că, cu excepția ei, numărul nu mai conține decât cifre impare și, cu siguranță, zerouri. În acest caz vom muta un zero pe ultima poziție.

Problema nu are soluție decât în situația în care toate cifrele numărului dat sunt impare.

În final vom scrie în fișierul de ieșire numărul obținut sau mesajul `NU EXISTA`.

Analiza complexității

Vom considera că numărul citit are k cifre; așadar, ordinul de complexitate al operației de citire a datelor de intrare este $O(k)$.

Operația de sortare a cifrelor poate fi realizată în timp liniar-logaritmic dar, datorită numărului mic de cifre, este suficientă folosirea unui algoritm de sortare care are ordinul de complexitate $O(k^2)$.

Identificarea și mutarea pe prima poziție a primei cifre diferite de zero se realizează în timp *liniar*.

Operația de identificare și mutare pe ultima poziție a cifrei pare se realizează tot în timp *liniar*.

Scrierea datelor de ieșire implică scrierea a k cifre, deci și această operație are ordinul de complexitate $O(k)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(k) + O(k^2) + O(k) + O(k) + O(k) = O(k^2)$.

Un număr N are întotdeauna $[\lg N] + 1$ cifre; așadar ordinul de complexitate al algoritmului, exprimat în funcție de N, este $O(\log^2 N)$.

Dificultate 10

Algoritmi	★☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	☆☆☆☆☆
Cunoștințe	★☆☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/numar.cpp

P020309: Suma

Vom calcula mai întâi suma S_0 a cifrelor numărului dat. Restul împărțirii la 10 al acestei sume este ultima cifră a sumei căutate. Câtul împărțirii la 10 al valorii S_0 este adunat la valoarea S_0, obținându-se valoarea S_1. Restul împărțirii la 10 al acestei sume este penultima cifră a sumei căutate. Câtul împărțirii la 10 al valorii S_1 este adunat la valoarea S_0, obținându-se valoarea S_2.

În general, câtul împărțirii la 10 a valorii S_i este adunat la valoarea S_0, obținându-se valoarea S_{i+1}. Restul împărțirii la 10 a unei valori S_i reprezintă o cifră a sumei căutate.

Dacă valoarea S_0 este mai mare decât 9 și numărul dat are k cifre, atunci rezultatul va avea $k + 1$ cifre; așadar, dacă ultima sumă S_{k-1} este mai mare decât 10, atunci câtul împărțirii sale la 10 este prima cifră a rezultatului.

Analiza complexității

Vom considera că numărul citit are k cifre; așadar, ordinul de complexitate al operației de citire a datelor de intrare este $O(k)$.

Determinarea sumei cifrelor numărului se realizează printr-o simplă parcurgere a acestora, deci ordinul de complexitate al acestei operații este $O(k)$.

Determinarea cifrelor rezultatului se realizează tot printr-o singură parcurgere, deci și această operație are ordinul de complexitate $O(k)$.

Scrierea datelor de ieșire implică scrierea a k cifre; așadar și această operație are ordinul de complexitate $O(k)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(k) + O(k) + O(k) + O(k) + O(k) = O(k)$.

Un număr N are întotdeauna $[\lg N] + 1$ cifre; așadar ordinul de complexitate al algoritmului, exprimat în funcție de N, este $O(\log N)$.

Dificultate 06

Algoritmi	☆☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	☆☆☆☆☆
Cunoștințe	★☆☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/suma.cpp

P020310: Numere

Definirea recursivă a problemei ilustrează foarte clar modul de rezolvare a acesteia. Vom crea două rutine recursive, una care generează numerele din prima jumătate a șirului și una care generează numerele din a doua jumătate a șirului.

Pentru prima parte a șirului, la fiecare pas i se vor genera numere cuprinse între 2^i și $3 \cdot 2^{i-1} - 1$. Pentru a doua parte a șirului, la fiecare pas i se vor genera numere cuprinse între $3 \cdot 2^{i-1}$ și $2^{i+1} - 1$.

Diferența principală a celor două rutine recursive este poziția apelului recursiv față de generarea elementelor. Pentru prima parte a șirului, apelul va apărea după generare, iar pentru a doua parte a șirului, apelul va apărea înaintea generării.

Analiza complexității

Datele de intrare constau într-un singur număr, deci ordinul de complexitate al operației de citire este $O(1)$.

Întotdeauna vor fi generate 2^N numere, deci ordinul de complexitate al operației de generare a acestora este $O(2^N)$.

Scrierea datelor în fișierul de ieșire este realizată pe parcursul determinării acestora, deci operația de scriere nu consumă timp suplimentar.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1) + O(2^N) = O(2^N)$.

Dificultate 08

Algoritmi	★☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	★☆☆☆☆
Cunoștințe	★☆☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/numere.cpp

P020311: Triunghi

Pentru fiecare punct dat va trebui să verificăm dacă se află sau nu în interiorul triunghiului considerat și să numărăm punctele din interiorul triunghiului.

O variantă mai simplă de rezolvare se bazează pe faptul că dacă un punct se află în interiorul triunghiului, pe una din laturi sau este un vârf, atunci suma ariilor celor trei triunghiuri determinate de punctul respectiv și perechi de vârfuri ale triunghiului dat este egală cu aria triunghiului dat (vezi figura 1).

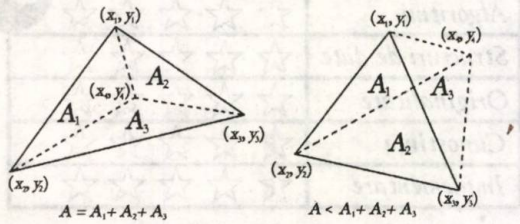


Figura 1

Așadar, pentru fiecare punct în parte vom determina ariile triunghiurilor determinate de punctul respectiv și două vârfuri ale triunghiului. Dacă suma celor trei arii este egală cu aria triunghiului, atunci punctul se află în interiorul triunghiului.

Se observă că dacă punctul este un vârf al triunghiului, două dintre cele trei arii vor fi egale cu 0, iar cea de-a treia arie va fi chiar aria triunghiului dat.

În cazul în care vârful se află pe una dintre laturi (dar nu este un vârf al triunghiului), atunci una dintre arii va fi egală cu 0.

Dacă se cunosc coordonatele vârfurilor unui triunghi, atunci formula de calcul a ariei acestuia este:

$$\left| \frac{1}{2} \cdot \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \right|.$$

În final vom scrie în fișierul de ieșire numărul punctelor aflate în interiorul triunghiului.

Analiza complexității

Citirea coordonatelor vârfurilor triunghiului constă în citirea a șase numere întregi, deci se realizează în timp *constant*.

Verificarea faptului dacă un punct se află sau nu în interiorul triunghiului considerat se realizează cu ajutorul unui număr constant de operații aritmetice. Această operație poate fi realizată pe măsura citirii coordonatelor punctelor. Așadar, ordinul de complexitate al operațiilor de citire a datelor și determinare a soluției este $O(N)$.

Datele de ieșire constau într-un singur număr, așadar ordinul de complexitate al operației de scriere a acestora este $O(1)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(1) + O(N) + O(1) = O(N)$.

Dificultate 12

Algoritmi	★☆☆☆☆
Structuri de date	☆☆☆☆☆
Originalitate	★☆☆☆☆
Cunoștințe	★☆☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/triunghi.cpp

P020312: Găuri

Vom determina toate regiunile formate din zerouri și vom eticheta elementele acestor regiuni folosind numere naturale, începând cu 1.

Pentru fiecare regiune determinată vom păstra dimensiunile acestora (numărul de elemente) într-un vector ai cărui indici sunt numerele folosite pentru etichetarea dimensiunilor regiunilor. Pentru a determina aceste regiuni vom folosi un simplu algoritm de umplere.

O gaură poate fi privită ca fiind o regiune formată din zerouri care nu conține nici un element aflat pe marginea matricei. Vom parcurge cele patru margini ale matricei și vom seta la 0 dimensiunile tuturor regiunilor care conțin elemente de pe aceste margini. Astfel, practic am eliminat toate regiunile care nu sunt găuri.

În final vom parcurge vectorul dimensiunilor și vom determina elementul care are valoarea maximă. Valoarea acestui element va fi scrisă în fișierul de ieșire.

Analiza complexității

Datele de intrare constau în dimensiunile matricei și valorile tuturor elementelor acesteia. Așadar ordinul de complexitate al operației de citire este $O(M \cdot N)$.

Operația de determinare a regiunilor formate din zerouri folosind un algoritm de umplere are ordinul de complexitate $O(M \cdot N)$.

Pentru a elimina regiunile care nu sunt găuri vom parcurge cele patru margini ale matricei, așadar această operația are ordinul de complexitate $O(M) + O(N) + O(M) + O(N) = O(M + N)$.

În total pot exista cel mult $[M \cdot N / 2] + 1$ regiuni (dacă matricea are aspectul unei table de șah). Așadar, operația de determinare a dimensiunii celei mai mari regiuni are ordinul de complexitate $O(M \cdot N)$.

Datele de ieșire constau într-un singur număr, operația de scriere a acestuia având ordinul de complexitate $O(1)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(M \cdot N) + O(M \cdot N) + O(M + N) + O(M \cdot N) + O(1) = O(M \cdot N)$.

	Dificultate 18
Algoritmi	★ ☆ ☆ ☆ ☆
Structuri de date	★ ☆ ☆ ☆ ☆
Originalitate	★ ☆ ☆ ☆ ☆
Cunoștințe	★ ★ ☆ ☆ ☆
Implementare	★ ☆ ☆ ☆ ☆

www.ginfo.ro/revista/13_3/surse/gauri.cpp

P020313: Exponent

Pentru a determina valoarea k, va trebui, mai întâi să descompunem numărul n în factori primi.

Pentru fiecare factor p care apare în descompunerea lui n, va trebui să determinăm puterea la care apare acest factor în descompunerea în factori primi a valorii $m!$.

Vom considera că p apare în descompunerea în factori primi a numărului n de u ori; cu alte cuvinte n poate fi scris sub forma $n = p^u \cdot q$, unde q este un număr întreg.

Vom determina acum puterea v la care apare p în descompunerea în factori primi a numărului $m!$. Valoarea v poate fi obținută astfel:

- pentru fiecare multiplu al lui p mai mic sau egal cu m, numărul p apare o dată în descompunerea lui $m!$;
- pentru fiecare multiplu al lui p^2 mai mic sau egal cu m, numărul p apare încă o dată în descompunerea lui $m!$;
- în general, pentru fiecare multiplu al lui p^i mai mic sau egal cu m, numărul p apare încă o dată în descompunerea lui $m!$.

Expresia *pentru fiecare multiplu al lui p^2 mai mic sau egal cu m* este echivalentă cu expresia *pentru fiecare multiplu al lui p mai mic sau egal cu $[m / p]$*.

În general, expresia *pentru fiecare multiplu al lui p^i mai mic sau egal cu m* este echivalentă cu expresia *pentru fiecare multiplu al lui p mai mic sau egal cu $[m / p^{i-1}]$*.

Ca urmare, valoarea v la care apare p în descompunerea în factori primi a numărului $m!$ poate fi determinată folosind următorul algoritm:

```
v ← 0
cât timp m > 0 execută
   v ← v + [m / p]
   m ← [m / p]
sfârșit cât timp
```

În aceste condiții, pentru ca $n^k = p^{u \cdot k} \cdot q^k$ să fie divizor al numărului $m!$, trebuie să avem $k \le [v / u]$.

Vom determina valorile maxime ale numărului k pentru toți divizorii p și o vom alege pe cea mai mică dintre acestea.

Analiza complexității

Intrarea constă în citirea a două numere, așadar ordinul de complexitate al operației de citire a datelor este $O(1)$.

Operația de descompunere în factori primi a unui număr natural n are ordinul de complexitate $O(\sqrt{n})$.

Numărul factorilor primi ai valorii n poate fi cel mult $[\log_2 n]$. Pentru un număr m dat, folosirea algoritmului prezentat pentru determinarea puterii la care apare p în descompunerea în factori primi a valorii $m!$ necesită parcurgerea a $[\log_p m]$ pași.

Ca urmare, operația de determinare a valorii k are ordinul de complexitate $O(\log m) \cdot O(\log n) = O(\log m \cdot \log n)$.

Ieșirea constă într-un singur număr, așadar ordinul de complexitate al operației de scriere a datelor este $O(1)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(1) + O(\sqrt{n}) + O(\log m \cdot \log n) + O(1) = O(\sqrt{n} + \log m \cdot \log n)$.

	Dificultate 18
Algoritmi	★ ☆ ☆ ☆ ☆
Structuri de date	☆ ☆ ☆ ☆ ☆
Originalitate	★ ★ ☆ ☆ ☆
Cunoștințe	★ ★ ☆ ☆ ☆
Implementare	★ ☆ ☆ ☆ ☆

www.ginfo.ro/revista/13_3/surse/expo.cpp

P020314: Bastioane

Se observă că pentru primul bastion va trebui să alegem o submulțime a soldaților care pot duce un număr total de pachete divizibil cu valoarea n. Pentru al doilea bastion, dintre cei k soldați rămași va trebui să alegem o submulțime a soldaților care pot duce un număr total de pachete divizibil cu valoarea k. Așadar, pentru al doilea bastion avem aceeași problemă pentru o valoarea k strict mai mică decât n.

Practic, la fiecare pas vom rezolva aceeași problemă pentru valori din ce în ce mai mici până în momentul în care nu mai există nici un soldat.

Așadar, la fiecare pas va trebui să determinăm o submulțime a unei mulțimi cu k elemente a cărei sumă să fie divizibilă cu k. În acest scop vom folosi principiul cutiei lui *Dirichlet*.

Presupunem că, la un moment dat, cei k soldați rămași pot duce $a_1, a_2, \ldots$, respectiv a_k pachete. Vom calcula sumele $S_i = a_1 + a_2 + \ldots + a_i$ folosind formula recurentă $S_1 = a_1$ și $S_i = a_i + S_{i-1}$ pentru toate valorile i cuprinse între 2 și k.

Potrivit principiului cutiei lui *Dirichlet*, dacă resturile împărțirii valorilor S_i la k sunt distincte, atunci unul dintre aceste resturi este 0.

Potrivit aceluiași principiu, dacă resturile nu sunt distincte, atunci două dintre acestea sunt egale. Dacă resturile împărțirii valorilor S_u și S_v sunt egale, atunci restul împărțirii valorii $S_v - S_u$ la k este 0. Așadar, valoarea $a_{u+1} + a_{u+2} + \ldots + a_v$ va fi divizibilă cu k.

În concluzie, va exista întotdeauna posibilitatea de a alege o submulțime a cărei sumă să fie divizibilă cu k. După alegerea acestei submulțimi, dacă ea nu conține toate cele k elemente, va trebui să rezolvăm aceeași problemă pentru o valoarea strict mai mică decât k (elementele mulțimii respective sunt eliminate).

Analiza complexității

Intrarea constă în citirea valorilor corespunzătoare celor n soldați, deci ordinul de complexitate al operației de citire a datelor este $O(n)$.

La fiecare pas va trebui să determinăm sumele S_i, operație care are ordinul de complexitate $O(k)$, unde $k = O(n)$ este numărul soldaților rămași la pasul respectiv.

Pe parcursul determinării sumelor vom păstra și resturile împărțirii lor la k, așadar vom putea identifica elementele care vor fi eliminate pe parcursul determinării sumelor. Eliminarea propriu-zisă este realizată în timp liniar. În cazul cel mai defavorabil la fiecare pas vom elimina un singur element, așadar putem avea cel mult n pași. Ca urmare, ordinul de complexitate al operației de repartizare a soldaților este $O(n) \cdot O(n) = O(n^2)$.

Datele de ieșire sunt scrise pe parcursul determinării soluțiilor, deci operația de scriere a acestora nu consumă timp suplimentar.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(n) + O(n^2) = O(n^2)$.

Dificultate **18**

Algoritmi	★☆☆☆☆
Structuri de date	★☆☆☆☆
Originalitate	★☆☆☆☆
Cunoștințe	★★☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/soldati.cpp

P020315: Graf

Pentru a determina cel mai îndepărtat nod față de cel identificat prin valoarea 1 va trebui să realizăm o parcurgere în în lățime (*Breadth First - BF*) a grafului.

O astfel de parcurgere implică atingerea nodurilor mai apropiate de nodul 1 înaintea atingerii nodurilor mai îndepărtate.

Ca urmare, ultimul nod atins de o astfel de parcurgere va fi întotdeauna unul dintre cele mai îndepărtate noduri, chiar dacă există și alte noduri aflate la aceeași distanță.

După determinarea ultimului nod parcurs vom scrie în fișierul de ieșire numărul său de ordine.

Analiza complexității

Intrarea constă în citirea informațiilor referitoare la cele M muchii ale grafului, deci ordinul de complexitate al operației de citire a datelor este $O(m)$. Pe parcursul citirii este construită și structura de date în care va fi păstrat graful, deci nu este consumat timp suplimentar pentru generarea acesteia.

Operația de parcurgerea în lățime a unui graf are ordinul de complexitate $O(m)$.

Datele de ieșire constau într-un singur număr, așadar ordinul de complexitate al operației de scriere este $O(1)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(m) + O(m) + O(1) = O(m)$.

Dificultate **20**

Algoritmi	★☆☆☆☆
Structuri de date	★☆☆☆☆
Originalitate	★☆☆☆☆
Cunoștințe	★★☆☆☆
Implementare	★☆☆☆☆

www.ginfo.ro/revista/13_3/surse/graf.cpp

Punctajele GInfo

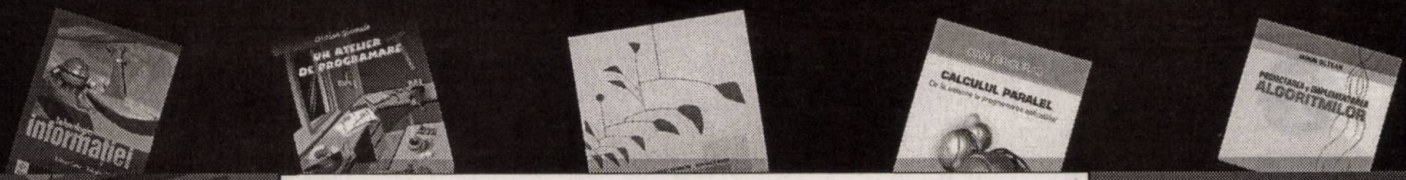
Mai mulți cititori și-au exprimat dorința de a afla modul în care sunt acordate punctajele de către redacția GInfo în momentul prezentării soluției unei probleme.

După cum probabil v-ați dat seama, sunt luate în considerare cinci criterii și anume: dificultatea algoritmilor folosiți, complexitatea structurilor de date utilizate, originalitatea problemei, cunoștințele teoretice necesare rezolvării problemei și dificultățile întâlnite în timpul implementării propriu-zise.

Pentru fiecare dintre aceste criterii este acordat un punctaj cuprins între 0 și 10 în funcție de diverse aspecte pe care nu le putem prezenta din motive de spațiu.

Pentru fiecare criteriu punctajul este un număr real care va fi rotunjit la cel mai apropiat număr întreg. Numărul steluțelor acordate este obținut prin împărțirea la 2 a acestui număr (așadar pentru fiecare "unitate" se acordă o "jumătate de steluță"). În calculul punctajului final se iau în considerare valorile reale, nu cele rotunjite. Formula de calcul este următoarea:

Dificultate = Algoritmi · 3 + Structuri de date · 2,5 + + Originalitate + Cunoștințe · 2,5 + + Implementare



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Bursele Agora

Vă prezentăm în continuare enunțurile problemelor propuse spre rezolvare la etapele #13-#16 ale celei de-a patra ediții a concursului de programare Bursele Agora.

P030311: Grupuri

Se consideră N numere întregi care trebuie repartizate în p grupuri. Grupurile sunt identificate prin numere naturale cuprinse între 1 și p. Repartizarea în grupuri trebuie să se realizeze astfel încât suma numerelor din oricare grup i să fie divizibilă cu numărul total de numere care fac parte din grupurile identificate prin valori cuprinse între i și p.

Date de intrare

Prima linie a fișierului de intrare **GROUPS.IN** conține numărul N al valorilor care trebuie repartizate în grupuri. Cea de-a doua linie va conține cele N numere, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **GROUPS.OUT** trebuie să conțină un număr de linii egal cu numărul grupurilor. Prima linie corespunde primului grup, a doua linie celui de-al doilea grup etc. Linia corespunzătoare unui grup va conține numerele care fac parte din grupul respectiv, separate prin câte un spațiu.

Restricții și precizări

- $1 \leq N \leq 1000$;
- cele N numere sunt cuprinse între 1 și 1000;
- numărul grupurilor nu este cunoscut; el va trebui determinat de program;
- fiecare grup va conține cel puțin un număr;
- dacă există mai multe soluții va fi generată doar una dintre ele;
- va exista întotdeauna cel puțin o soluție.

Exemplu

GROUPS.IN	GROUPS.OUT
6	2 4
4 10 3 9 2 3	3 9
	10
	3

Explicație

Sunt formate patru grupuri dintre care primele două conțin câte două numere, iar ultimele două conțin unul singur.

Suma numerelor din primul grup este 2 + 4 = 6 care este divizibilă cu 6 (numărul elementelor din grupurile identificate prin 1, 2, 3 și 4). Suma numerelor din al doilea grup este 3 + 9 = 12 care este divizibilă cu 4 (numărul elementelor din grupurile identificate prin 2, 3 și 4). Al treilea grup este format doar din numărul 10; acesta este divizibil cu 2 (numărul elementelor din grupurile identificate prin 3 și 4). Al patrulea grup este format doar din numărul 3; acesta este divizibil cu 1 (numărul elementelor din grupul identificat prin 4). Există mai multe alte soluții alternative; una dintre ele este:

```
2 3 9 10
4
3
```

Timp de execuție: 1 secundă / test

P030312: Războiul stelelor

Cu mult timp în urmă, într-o galaxie foarte îndepărtată, s-a desfășurat un *Consiliu* al *Cavalerilor Jedi* în cadrul căruia s-a discutat problema noii amenințări reprezentate de partea întunecată a *Forței*. În ultima perioadă au fost semnalate numeroase apariții ale *Lorzilor Sith*, motiv pentru care *Consiliul* a trebuit să ia o decizie importantă. După luarea deciziei *Yoda* a ieșit din *Sala de Consiliu* și s-a îndreptat direct spre *Obi Wan* care aștepta cu nerăbdare să afle decizia luată.

Yoda: Consiliul decis este!
Obi Wan: Pari cam nemulțumit, Maestre Yoda.
Yoda: În dezacord cu decizia eu sunt!
Obi Wan: Dar ce s-a hotărât până la urmă?
Yoda: Mulți Cavaleri pe planete îndepărtate trimiși trebuie!
Obi Wan: De ce?
Yoda: Găuri de vierme descoperite au fost! Pentru deplasare rapidă între planete găurile folosite pot fi!
Obi Wan: Și care este problema?
Yoda: Lorzii Sith găuri folosesc pentru scopurile lor mârșave!
Obi Wan: Se cunosc planetele legate prin găuri de vierme?

Yoda: Amplasamentul tuturor intrărilor și ieșirilor găurilor Consiliul îl cunoaște!
Obi Wan: Și ce e de făcut acum?
Yoda: Cavaleri Jedi pe fiecare planetă lângă care se află găuri trimiși vor fi!
Obi Wan: Deci va fi trimis câte un Jedi pe fiecare dintre aceste planete?
Yoda: Mult mai complicată problema este! Numărul Cavalerilor Jedi de pe o planetă diferit să fie trebuie de numărul Cavalerilor de pe oricare planetă legată printr-o gaură de planeta aceea!
Obi Wan: De ce?
Yoda: Forța în acest fel ne-a sfătuit!
Obi Wan: Atunci cred că ar trebui să facem în așa fel încât să trimitem cât mai puțini Jedi!
Yoda: Sfatul tău bun este! Așa vom face!

Așadar, *Consiliul Cavalerilor Jedi* va trebui să trimită *Jedi* pe fiecare planetă lângă care a fost descoperită o gaură de vierme, iar numărul *Cavalerilor Jedi* de pe oricare dintre planete trebuie să fie diferit de cel al *Cavalerilor* trimiși pe oricare dintre planetele învecinate (legate de planeta respectivă printr-o gaură de vierme). În plus, numărul total al *Cavalerilor Jedi* trimiși va trebui să fie cât mai mic posibil.

Date de intrare

Prima linie a fișierului de intrare **STARWARS.IN** conține numărul N al planetelor lângă care au fost descoperite găuri de vierme și numărul total M al găurilor descoperite.

Fiecare dintre următoarele M linii va conține câte două numere x și y, separate printr-un spațiu, cu semnificația: *există o gaură de vierme între planeta identificată prin numărul x și planeta identificată prin numărul y.*

Date de ieșire

Fișierul de ieșire **STARWARS.OUT** va conține N linii. Cea de-a i-a linie va conține numărul *Cavalerilor Jedi* trimiși pe planeta identificată prin numărul i.

Restricții și precizări

- $1 \le N \le 100$;
- $1 \le M \le 1000$;
- planetele sunt identificate prin numere întregi cuprinse între 1 și N;
- printr-o gaură de vierme se poate circula în ambele sensuri;
- între două planete poate exista cel mult o gaură de vierme.

Exemplu

STARWARS.IN	STARWARS.OUT
5 4	2
1 2	1
1 3	1
1 4	1
1 5	1

Modalitatea de acordare a punctajului

Vom considera că pentru fiecare test, se vor putea obține cel mult X puncte. Concurenții care vor obține cea mai mică valoare *NrMin* pentru numărul total al *Cavalerilor Jedi* vor primi X puncte pentru testul respectiv.

Ceilalți concurenți, care au stabilit o repartizare corectă a *Cavalerilor Jedi*, dar numărul total al acestora este *Nr* > *NrMin*, vor obține $X \cdot NrMin / Nr$ puncte pentru testul respectiv. Această valoare va fi aproximată cu două zecimale exacte.

Punctajul final va fi obținut prin adunarea punctajelor de la fiecare test și rotunjirea acestuia la cel mai apropiat număr întreg.

Dacă repartizarea nu respectă condițiile impuse, concurenții nu vor primi nici un punct pentru testul respectiv.

De exemplu, dacă pentru un test se pot obține cel mult 5 puncte, cel mai bun rezultat obținut de un concurent constă într-o repartizare în care sunt folosiți 24 de *Cavaleri Jedi*, iar un alt concurent obține o soluție în care este nevoie de 57 de *Cavaleri Jedi*, atunci, pentru testul respectiv, punctajul concurentului va fi de $5 \cdot 24 / 57 = 2.10$ puncte.

Timp de execuție: 1 secundă / test

P030313: Galaxie

Confederația Galactică deține controlul asupra a N planete între care există un număr total de M rute comerciale directe. Planetele vor fi identificate prin numere cuprinse între 1 și N, iar rutele comerciale prin numere cuprinse între 1 și M.

Fiecare rută comercială directă este folosită pentru transporturi între perechi distincte de planete. Un circuit comercial este format din k planete distincte $a_1, a_2, ..., a_k$ astfel încât există o rută comercială directă între planetele a_i și a_{i+1} pentru i cuprins între 1 și k - 1 și există o rută comercială directă între planetele a_k și a_1.

Două circuite comerciale sunt distincte dacă și numai dacă există cel puțin o rută comercială care apare în unul dintre ele și nu apare în celălalt. Pentru fiecare rută comercială, va trebui să stabiliți din câte circuite comerciale distincte face parte.

Date de intrare

Fișierele de intrare sunt deschise și pot fi descărcate folosind *link*-urile de pe pagina dedicată problemei; adresa acesteia este **http://www.ginfo.ro/concurs/runda15/index.shtml**. Numele fișierelor au forma **GALAXYXX.IN**, unde **XX** ia valori între **01** și **20** și reprezintă numărul testului.

Fiecare dintre cele 20 de fișiere de intrare conține pe prima linie numărul N al planetelor și numărul M al rutelor comerciale directe. Următoarele M linii vor conține câte două numere întregi distincte cuprinse între 1 și N care sunt numerele de identificare ale planetelor legate între ele prin ruta comercială corespunzătoare liniei. Numerele de pe o linie sunt separate prin câte un spațiu.

Date de ieșire

Pentru această problemă nu va trebui să scrieți un program, ci să generați doar cele 20 de fișiere de ieșire corecte. Acestea vor fi denumite **GALAXYXX.OUT**, unde **XX** ia valori între 01 și 20 și reprezintă numărul testului.

Prima linie a fișierului de ieșire va conține textul GALAXY - TEST #XX, unde XX reprezintă numărul testului. Următoarele *M* linii vor conține câte un număr întreg care reprezintă numărul de circuite comerciale distincte din care face parte ruta corespunzătoare liniei. Informațiile referitoare la rute vor apărea în ordinea în care sunt descrise rutele în fișierul de intrare corespunzător.

Exemplu

Vom considera că acest exemplu reprezintă testul **00**.

GALAXY00.IN	**GALAXY00.OUT**
4 5	GALAXY - TEST #00
1 2	2
1 3	2
2 3	2
2 4	2
3 4	2

P030314: Săgeți

Mickey și *Minnie* au ajuns în fața unui perete pe care erau desenate mai multe cercuri și mai multe săgeți, fiecare săgeată pornind dintr-un cerc și ajungând la un alt cerc.

Desenul era foarte complicat, dar *Minnie* l-a întrebat pe *Mickey* dacă există vreun cerc în care intră săgeți care pornesc de la toate celelalte cercuri, dar din care nu pleacă nici o săgeată.

Pentru a o impresiona pe *Minnie*, *Mickey* a spus că va studia problema, dar se pare că nu se prea descurcă, motiv pentru care vă cere ajutorul.

Din nefericire, nu puteți comunica decât prin intermediul unui telefon. *Mickey* vă spune la început numărul *N* al cercurilor, dar nu are răbdare să vă descrie toate săgețile.

Datorită nerăbdării lui *Mickey*, îl veți putea întreba doar de $2 \cdot N$ ori dacă există o săgeată care pleacă de la un anumit cerc și ajunge la un anumit cerc. Din fericire, ați auzit-o la telefon pe *Minnie* strigând "*L-am găsit*", motiv pentru care știți cu siguranță că există un astfel de cerc.

Pentru a simula convorbirea cu *Mickey* aveți la dispoziție o bibliotecă externă care vă pune la dispoziție rutine pentru:

- inițializare;
- determinarea numărului de cercuri;
- verificarea existenței unei săgeți care pornește de la un anumit cerc *x* și ajunge la un alt cerc *y*;
- furnizarea cercului în care intră săgeți de la toate celelalte cercuri și din care nu pleacă nici o săgeată.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul dumneavoastră instrucțiunea **uses** arrows; pentru limbajul *Pascal* și `#include "arrows.h"` pentru limbajul *C/C++*. În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

procedure Init;
void Init()
- trebuie apelată la începutul programului și este folosită de către bibliotecă pentru inițializare;
- întrerupe execuția programului dacă este apelată a doua oară.

function GetN:Integer;
int GetN()
- returnează numărul de cercuri de pe perete.

function IsArrow(x,y:Integer):Boolean;
int IsArrow(**int** x,**int** z)
- returnează true (valoare nenulă) dacă există o săgeată care pornește de la cercul identificat prin *x* și ajunge la cercul identificat prin *y* și false (0) în caz contrar;
- întrerupe execuția programului dacă cel puțin una dintre valorile i și j nu este un număr întreg cuprins între 1 și *N* sau dacă este apelată a $2 \cdot N + 1$ - a oară.

procedure Result(circle:Integer);
void Result(**int** circle)
- acceptă ca rezultat faptul că circle este cercul căutat;
- validează rezultatul;
- întrerupe execuția programului.

Toate funcțiile și procedurile (excepție: Init) întrerup execuția programului dacă sunt apelate înaintea inițializării.

Restricții și precizări

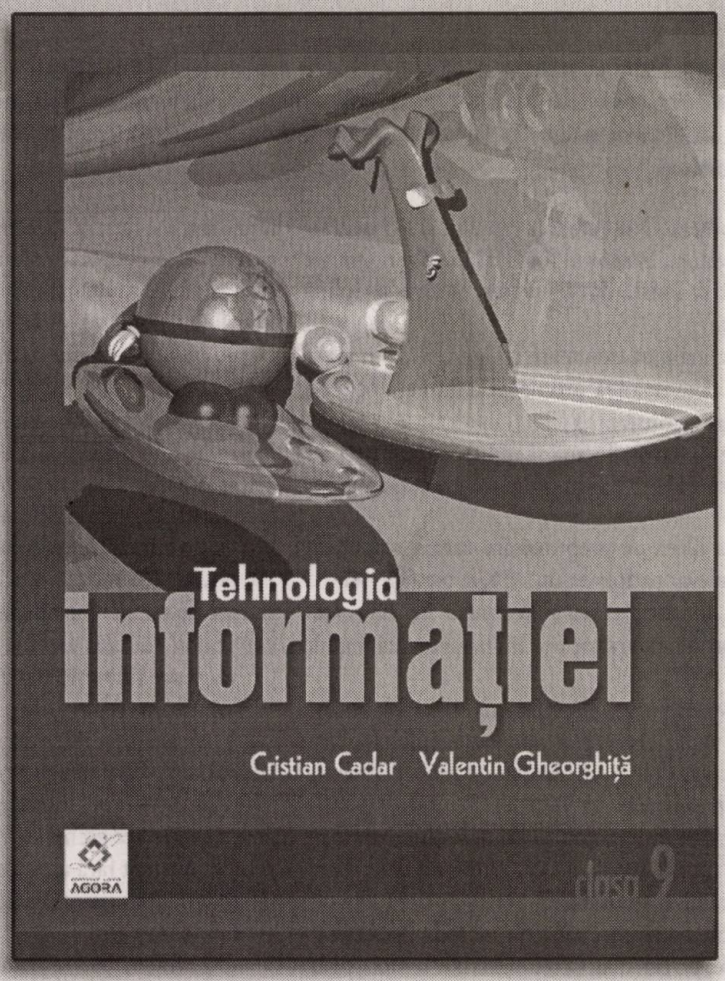
- $1 \le N \le 100$;
- programul vostru nu va citi date din nici un fișier de intrare și nu va scrie date în nici un fișier de ieșire;
- cercurile sunt identificate prin numere cuprinse între 1 și *N*;
- există cel mult o săgeată de la un cerc *x* la un cerc *y*;
- nu există săgeți care pleacă de la un cerc *x* și ajung la același cerc *x*;
- dacă există o săgeată de la un cerc *x* la un cerc *y*, atunci este posibil să existe și o săgeată de la cercul *y* la cercul *x*.

Exemple de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
Init	-
GetN	3
IsArrow(1,2)	true (1)
IsArrow(1,3)	true (1)
IsArrow(2,3)	true (1)
IsArrow(3,1)	false (0)
IsArrow(2,1)	true (1)
IsArrow(3,2)	false (0)
Result(3)	-

Timp de execuție: 1 secundă / test

COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel.0264 - 192422

COMPUTER LIBRIS AGORA

PREPROCESAREA

Mihai Stroe

Articolul de față prezintă o categorie de optimizări importante în rezolvarea problemelor: soluțiile bazate pe preprocesare. Vom prezenta modul în care poate fi utilizată această tehnică pentru a îmbunătăți timpul de execuție al programelor.

Termenul de *preprocesare* se referă la calculul și stocarea în memorie a unor valori, în scopul folosirii lor ulterioare pe parcursul rezolvării. Există două niveluri la care se aplică aceste optimizări:

- calcule efectuate înaintea executării programului; în acest caz, programul care rezolvă problema conține tabele cu valori precalculate (constante);
- calcule realizate înaintea efectuării unor anumiți pași de rezolvare, pentru reducerea timpului de execuție a acestora.

Vom prezenta ideea de preprocesare care a apărut, în contextul olimpiadelor naționale, în 1996, la barajele pentru selectarea echipei lotului pentru *Balcaniada de Informatică*. Una dintre problemele propuse era următoarea (enunț ușor modificat):

"Să se amplaseze pe muchiile unui cub numerele de la 1 la 12, astfel încât suma numerelor de pe muchiile care formează fiecare față să fie egală. Problema nu are date de intrare; se cer toate soluțiile."

Problema stabilea, prin enunț, o numerotare a muchiilor și amplasarea numărului 1 pe o anumită muchie. Timpul de execuție era de o secundă.

Evident, problema se rezolvă prin metoda *backtracking*. Din nefericire, pe calculatoarele existente la momentul respectiv, rezolvări nu foarte eficient implementate depășeau cu puțin timpul limită.

În aceste condiții, o parte dintre concurenți au aplicat următoarea tehnică:

- se rulează programul în timpul probei, obținându-se soluțiile într-un fișier;
- se observă că numărul de soluții este mic (80);
- se transformă fișierul rezultat într-un program care afișează cele 80 de soluții.

Cel mai simplu mod de a realiza transformarea constă în introducerea soluțiilor între ghilimele, prefixarea lor cu un `"writeln("` și finalizarea cu șirul `");"` (în *Pascal*). În acest mod, programul putea fi completat cu un **`"begin"`** și un **`"end."`** și ar fi rezultat un program *Pascal* corect care rezolvă problema (de fapt, afișarea se realiza într-un fișier; am ales această modalitate de prezentare pentru a simplifica înțelegerea soluției). Ideea este de a realiza afișarea, în programul care generează soluțiile, într-un mod care să ușureze cât mai mult scrierea programului care le va folosi. După cum se va vedea în continuare, pentru anumite probleme folosirea poate consta în mai mult decât tipărirea unor valori.

Bineînțeles, la corectare au apărut discuții legate de valoarea acestei rezolvări. Deoarece regulamentul nu era încălcat, s-a decis ca astfel de soluții să fie acceptate, regulă care a rămas valabilă pentru viitor. Altfel spus, programele bazate pe precalcularea soluțiilor sau a unor date ajutătoare nu pot fi depunctate; corectarea se realizează numai în funcție de rezultatele obținute la teste, fără a se ține cont de conținutul codului sursă.

Ca toate lucrurile bune care apar în contextul concursurilor (reamintim, printre altele, *backtracking*-ul *"omorât"*), această metodă a fost preluată și folosită din plin la concursurile care au urmat, inclusiv în cadrul concursurilor *"Bursele AGORA"*. În prezent, cunoașterea acestei metode este considerată esențială, iar anumite probleme nu se pot rezolva eficient fără ajutorul ei.

Vom continua prin a prezenta câteva abordări din cea de-a doua categorie.

O problemă interesantă, datorită optimizărilor care pot fi introduse, este următoarea:

"Se dau N puncte în plan de coordonate întregi ($3 \le N \le 100$). În aceste puncte locuiesc oameni. În două din aceste puncte se pot deschide restaurante. Fiecare om va mânca la restaurantul cel mai apropiat. Să se aleagă locațiile în care se vor deschide restaurante, astfel încât distanța totală parcursă de oameni de la casele lor până la restaurante să fie minimă. Oamenii care au restaurante acasă parcurg distanța 0."

Limitele problemei conduc la concluzia că rezolvarea prin metoda forței brute este acceptabilă. Să o examinăm:

distanța optimă devine `+infinit`
`pentru` *fiecare punct* `i` *între* `1` *și* `N` *(ales ca prim restaurant)* **`execută`**
  `pentru` *fiecare punct* `j` *între* `1` *și* N *(ales ca al doilea restaurant)* **`execută`**

distanța totală devine 0
pentru *fiecare punct* K *între* 1 *și* N **execută**
d1 *devine distanța de la* i *la* K (*)
d2 *devine distanța de la* j *la* K (*)
min *devine minimul dintre* d1 *și* d2
min *se adună la distanța totală*
distanța totală este comparată cu distanța optimă și, dacă este cazul, o actualizăm pe aceasta.

Complexitatea acestei metode este $O(N^3)$, deci timpul de rulare ar trebui să fie rezonabil.

Cum implementăm pașii marcați cu (*)? Prima variantă constă în realizarea unei funcții `dist(i,j)` și apelarea ei:

```
function dist(i,j:integer):real;
begin
  dist:=sqrt(sqr(x[i] - x[j]) + sqr(y[i] - y[j]));
end;
```

Evident d1 devine `dist(i,K)` și d2 devine `dist(j,K)`.

Se efectuează $O(N^3)$ apeluri de funcție, ceea ce duce la performanțe slabe.

O variantă ușor îmbunătățită constă în calcularea distanțelor în cadrul programului, fără a scrie o funcție separată:

```
d1:=sqrt(sqr(x[i] - x[K]) + sqr(y[i] - y[K]));
d2:=sqrt(sqr(x[j] - x[K]) + sqr(y[j] - y[K]));
```

Nu se mai efectuează apeluri de funcții `dist`, dar, pe cazul cel mai defavorabil, se calculează aproximativ un milion de distanțe (ceea ce înseamnă un milion de extrageri de rădăcini pătrate). Soluția bazată pe preprocesare constă în definirea unei matrice `dist`:

```
var dist:array[1..100,1..100] of real;
```

Matricea va fi completată la începutul programului, imediat după citirea datelor:

```
cor i:=1 to n do
  cor j:=i + 1 to n do begin
    dist[i,j]:=sqrt(sqr(x[i] - x[j]) + sqr(y[i]
                                    - y[j]));
    dist[j,i]:=dist[i,j];
  end;
```

Din acest moment, distanțele între oricare două puncte sunt disponibile și pot fi folosite în orice moment prin citirea din tabelă. S-au efectuat aproximativ 5000 de operații de extragere a rădăcinii pătrate. În plus, s-au economisit înmulțiri și adresări în vectorii x și y. Complexitatea este aceeași; timpul de execuție este însă redus semnificativ.

După examinarea acestei soluții, sunt evidente alte tipuri de valori care pot fi precalculate: tabele de logaritmi, rădăcini pătrate, funcții trigonometrice, numere prime etc. De exemplu (revenind în contextul calculelor efectuate înaintea lansării în execuție), un program care va folosi intens primele 1000 de numere prime poate începe astfel:

```
const prime:array[1..1000] of longint
      = (2, 3, 5, 7, 11, 13, 17, ...);
```

Începutul programului poate fi obținut cu ajutorul unui alt program, care generează primele 1000 de numere prime și le afișează câte 20 pe linie (pentru a nu depăși numărul maxim de caractere pe linie permis), separate prin virgulă. Fișierul rezultat este apoi completat cu codul programului.

În acest caz, primele 1000 de numere prime ar fi putut fi calculate și la executarea programului; există însă contexte în care vectorii de constante sunt indispensabili.

O optimizare utilă în multe probleme constă în construirea vectorului / matricei sumelor parțiale. O vom ilustra cu ajutorul următoarelor două probleme:

- *Se consideră un vector D cu N ≤ 10000 elemente, numere întregi. Urmează K întrebări, K ≤ 10000. O întrebare constă în două numere întregi A și B, între 1 și N inclusiv, cu A ≤ B. Pentru fiecare întrebare afișați suma elementelor vectorului, situate între pozițiile A și B inclusiv.*
- *Se dă o matrice D cu M linii și N coloane (1 ≤ M, N ≤ 100). Urmează K întrebări, K ≤ 10000. O întrebare constă în 4 numere întregi L1, C1, L2, C2 cu 1 ≤ L1 ≤ L2 ≤ M și 1 ≤ C1 ≤ C2 ≤ N. Pentru fiecare întrebare, afișați suma elementelor din matrice, din dreptunghiul determinat de liniile L1 și L2 și coloanele C1 și C2.*

Metoda bazată pe parcurgerea vectorului / matricei este ineficientă și va obține foarte puține puncte. Există însă metode în care răspunsul la fiecare întrebare se calculează în timp constant, datorită unor preprocesări efectuate după citirea datelor.

Pentru prima problemă, se calculează vectorul S al sumelor parțiale, definit astfel:

```
S[0] = 0
S[1] = D[1]
S[2] = D[1]+D[2]
S[3] = D[1]+D[2]+D[3]
...
```

S[i] = *suma primelor* i *elemente din vectorul* D. *Evident,* S[i]=S[i-1]+D[i].

Calculul vectorului S se realizează în $O(N)$. După acest pas, la fiecare întrebare se răspunde scăzând `S[A - 1]` din `S[B]`. Astfel, `S[B]` este suma primelor B elemente, iar `S[A - 1]` este suma primelor A - 1; realizând scăderea prezentată anterior se obține suma elementelor situate între pozițiile A și B.

Pentru cazul matricei, calculele sunt puțin mai complicate. Definim `S[i, j]` ca fiind suma elementelor din dreptunghiul delimitat de liniile 1 și i și de coloanele 1 și j. Valorile din S se calculează unele din altele, astfel:

```
S[i, j]:=S[i-1, j]+S[i, j-1]-
S[i-1, j-1]+D[i, j].
```

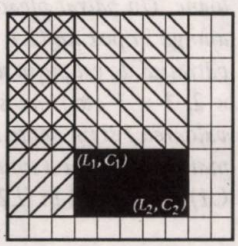


Figura 1

Cunoscând `S[i, j]`, suma elementelor din dreptunghiul determinat de liniile L1 și L2 și coloanele C1 și C2 se calculează astfel:

```
Suma:=S[L2, C2]-S[L1-1, C2]-S[L1, C2-1]
      +S[L1-1, C1-1].
```

Așadar răspunsul la o întrebare este calculat într-un timp de ordinul $O(1)$.

Calculul vectorului sumelor parțiale poate fi folosit și în unele probleme bazate pe matrice. Se pot calcula matricele sumelor parțiale pe linii, respectiv pe coloane. Mai exact, `LIN[i, j]` este suma primelor `j` elemente din linia `i`, iar `COL[i, j]` suma primelor `i` elemente din coloana `j`. Invităm cititorul să găsească o utilizare a acestor matrice.

Încheiem prezentarea cu câteva probleme rezolvabile cu ajutorul preprocesării, bazate pe noțiunile prezentate anterior.

Numere prime

"Se dau două numere A și B, cu 0 < A < B < 5000000. Afișați suma numerelor prime aflate în intervalul [A, B]. Timp de execuție: 1 secundă/test."

Un program care determină numerele prime dintre A și B și calculează suma lor nu se va încadra în timp pe testele în care intervalul este mare. Există deci o soluție alternativă.

Calculăm un vector S, unde $S[i]$ conține suma primelor $i \cdot 1000$ numere prime, unde i este cuprins între 1 și 5000. Pe baza acestui vector și a cunoștințelor despre metoda vectorului sumelor parțiale, prezentate anterior, răspundem la cerința problemei astfel:

- calculăm pozițiile X și Y, multipli de 1000, situate între A și B, cu X cât mai apropiat de A și Y cât mai apropiat de B. De exemplu, pentru $A = 53478$ și $B = 674876$, avem $X = 54000$ și $Y = 674000$;
- suma numerelor prime situate între X și Y este: $S[Y/1000] - S[X/1000 - 1]$;
- suma numerelor prime dintre A și B se obține folosind suma determinată mai sus, la care adunăm numerele prime dintre A și X și pe cele dintre Y și B.

Vectorul S este calculat înainte de executarea programului, cu ajutorul unui program auxiliar (care, evident, se va termina după mai mult de o secundă, dar timpul său de rulare nu contează la evaluare). Programul auxiliar scrie vectorul S la începutul unui fișier; sumele sunt puse câte 10 - 20 pe linie și separate prin virgule. Fișierul este completat cu programul care realizează citirea datelor de intrare și calculează răspunsul, conform metodei de mai sus.

Pătratul și labirintul

"Se dă o matrice cu elemente de 0 și 1, cu N linii și N coloane. Un pătrat pleacă din poziția (L1, C1) și trebuie să ajungă în poziția (L2, C2). Pătratul constă într-o matrice pătratică de cel puțin un element (deci poate fi un pătrat 2 × 2, 3 × 3 etc.), nu se poate deplasa peste elemente cu valoarea 1 și nu poate ieși din matrice. Să se determine cea mai mare latură a unui pătrat care poate să plece din (L1, C1) și să ajungă în (L2, C2), respectând restricțiile."

De exemplu, pentru matricea:

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 1
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0
```

și $L1 = 1$, $C1 = 1$, $L2 = 4$, $C2 = 4$, pătratul maximal are latura 2 și poate fi deplasat prin trei mutări în jos și trei mutări în dreapta.

Rezolvarea clasică a acestei probleme se bazează pe algoritmul lui *Lee*. Acesta (considerat cunoscut în cadrul expunerii de față) este realizat pentru un pătrat de latura 1 și se bazează pe introducerea într-o coadă a pozițiilor în care se poate afla pătratul, în ordinea distanței lor față de poziția inițială.

Pentru a adapta algoritmul la pătratul de dimensiune variabilă, putem alege următoarea soluție:

pentru *fiecare dimensiune* `D` *a pătratului* **execută**
  se plasează pătratul în poziția inițială (dacă este posibil);
  se încearcă deplasarea pătratului, conformă algoritmului lui Lee;
  dacă deplasarea nu a reușit, se afișează soluția de la pasul precedent și se termină execuția.

În cadrul algoritmului lui *Lee*, verificarea dacă pătratul se poate deplasa într-o poziție vecină se realizează în D pași, examinând cele D elemente noi care urmează a fi ocupate prin deplasarea pătratului (pentru ca aceasta să fie posibilă, toate trebuie să fie nule). Complexitatea pe cazul cel mai defavorabil este $O(N^4)$.

Prima îmbunătățire a programului constă în calculul valorii optime a lui D cu ajutorul căutării binare. Se încearcă deplasarea unui pătrat de dimensiune $N/2$, dacă aceasta a reușit se încearcă un pătrat mai mare, în caz contrar un pătrat mai mic etc. Complexitatea a fost redusă la $O(N^3 \cdot \log N)$.

Se observă că verificarea dacă pătratul poate fi deplasat într-o poziție vecină se poate realiza cu ajutorul preprocesării. Se calculează matricele LIN și COL amintite anterior:

- $LIN[i, j]$ este suma primelor j elemente din linia i;
- $COL[i, j]$ este suma primelor i elemente din coloana j.

În acest context, condiția pentru ca, de exemplu, un pătrat de dimensiune D, aflat în poziția (L, C), să poată fi deplasat cu o poziție mai sus, este ca diferența $LIN[L\text{-}1, C\text{+}D\text{-}1] - LIN[L\text{-}1, C\text{-}1]$ să fie 0. Diferența respectivă dă suma elementelor situate pe linia L-1, între coloanele C și C+D-1. Pot fi găsite condiții similare pentru deplasarea în jos, spre stânga și spre dreapta. Complexitatea noii rezolvări este $O(N^2 \cdot \log N)$. Lăsăm ca exercițiu pentru cititor încercarea rezolvării problemei cu o complexitate $O(N^2)$.

Concluzie

Preprocesarea este o tehnică utilă. Stăpânirea ei poate conduce în multe cazuri la reducerea timpului de execuție, prin evitarea recalculării unor valori. În alte cazuri, se poate ajunge și la scăderi importante ale complexității.

Conceptul de ALGORITM - abordare MODERNĂ

Marin Vlada

În cadrul acestui al doilea articol dedicat unei abordări moderne a conceptului de algoritm va fi prezentat detaliat modul în care ar trebui privit algoritmul luând în considerare diferite aspecte care sunt practic desconsiderate de abordarea clasică.

Practica rezolvării problemelor folosind un limbaj de programare a determinat de-a lungul timpului diverse abordări în funcție de performanța limbajului de programare, performanța calculatorului și, nu în ultimul rând, în funcție de metodele și tehnicile avansate privind implementarea raționamentelor pentru demonstrațiile corespunzătoare problemelor.

În [13] se arată că "*un limbaj de programare acționează ca o interfață între universul real al problemei de rezolvat și programul de rezolvare, punând la dispoziție o serie de elemente constructive (entități) și legi de combinare a acestora, prin care elementele problemei și acțiunile de rezolvare pot fi reprezentate și prelucrate la nivelul programului. A construi un program de rezolvare a problemei înseamnă, în esență, a găsi modalitatea de agregare a acestor entități, în așa fel încât rezultatul să constituie o replică a procesului pe care un rezolvitor uman l-ar executa pentru a rezolva problema*". Rezolvarea teoretică a unei probleme nu garantează și rezolvarea ei practică folosind calculatorul.

În general, un limbaj de programare este menit să faciliteze rezolvarea unor clase de probleme și se pretează mai bine anumitor *tipuri de algoritmi*. Este nevoie de experiență în utilizarea și cunoașterea calculatorului, de competență și intuiție, este nevoie de inspirație și creație. În astfel de situații este nevoie de cunoașterea mai multor limbaje de programare pentru a alege limbajul de programare adecvat pentru clasa de probleme din care face parte problema de rezolvat.

Experiența a arătat că, atunci când nu este ales limbajul de programare corespunzător, dacă totuși se ajunge să se rezolve problema, s-a făcut risipă de resurse (timp, memorie, finanțe etc.) și, prin urmare, eficiența și performanța au avut de suferit.

Conform [13], rezolvarea unei probleme folosind un limbaj de programare modern trebuie să urmeze următoarele etape:

- ***etapa de analiză și abstractizare a problemei*** care constă în identificarea *obiectelor* implicate în rezolvare și a *acțiunilor de transformare* corespunzătoare acestora. Un *obiect* este un element esențial al unei probleme, element care poate fi analizat independent, prelucrabil prin *operatori specifici*, în funcție de *tipul* său, și a cărui *stare curentă* influențează *starea problemei* din punct de vedere al rezolvării acesteia. Rezultatul acestei prime etape este *universul abstract al problemei* (*UAP*), care scoate în evidență *mulțimea tipurilor de obiecte*, *mulțimea obiectelor*, *mulțimea relațiilor între obiecte* și *mulțimea restricțiilor de prelucrare* necesare rezolvării problemei;
- ***etapa găsirii unei metode acceptabile de rezolvare*** care constă în precizarea exactă a *operatorilor de prelucrare* proprii obiectelor din *UAP*, *elaborării algoritmului de rezolvare* și, în final, *codificării algoritmului* și *reprezentării obiectelor* din *UAP* folosind entitățile și legile de combinare oferite de limbajul de programare.

Trebuie să precizăm faptul că aceste etape sunt influențate de natura și performanța limbajului de programare folosit, de entitățile prin care un program poate fi construit folosind limbajul ales.

În domeniul *inteligenței artificiale* este utilizat limbajul *Prolog* care oferă facilități de programare declarativă / obiectuală și care este un exemplu concludent de limbaj care impune parcurgerea celor două etape precizate mai

sus. De asemenea, limbajul C++ oferă facilități moderne pentru rezolvarea diverselor probleme, respectând principiile *OOP* ([28, 29, 30]).

De asemenea, în domeniul *graficii pe calculator*, aceste etape sunt frecvente. Limbajul **Java** este un limbaj modern care utilizează din plin conceptele *programării orientate obiect*.

Astăzi, perfomanța unui *informatician / programator* este determinată de experiența și competența obținute în desfășurarea celor două etape (***Analiză, Programare***):

- ***etapa gândirii obiectuale*** (***Analiză / Proiectare***) - modul de analiză și abstractizare a problemelor prin definirea corectă a obiectelor, a tipurilor de obiecte, a relațiilor dintre obiecte și a operatorilor specifici (*elaborarea UAP; etapa de concepție și analiză - proiectare*);
- ***etapa gândirii algoritmice*** (***Programare / Execuție***) - alegerea și aplicarea corectă a unor metode de rezolvare prin precizarea exactă a operatorilor de prelucrare a obiectelor, reprezentarea corectă a strategiilor algoritmice, reprezentarea codificată a obiectelor și a prelucrărilor conform unui limbaj de programare (*elaborarea algoritmului și programului; etapa de programare - codificare, implementare și execuție*).

În prezent este tot mai des invocată reprezentarea problemelor folosind concepte *OOP* (*Object Oriented Programming*). Conceptul de ***obiect*** (**M. Minsky**, *The Society of Mind*, Touchstone Books, New York, 1986) are un rol important în știința cunoașterii și educației. Un *obiect* modelează o *entitate* din *lumea reală sau virtuală / imaginară* [14]. Prin definirea obiectelor se modelează diferite tipuri de entități și concepte din lumea reală sau imaginară.

În *activitatea de rezolvare a problemelor* trebuie să se *identifice / definească* obiectele din cadrul problemelor care provin din diferite domenii: științifice, economice, sociale etc.

Identificarea obiectelor este echivalentă cu determinarea entităților și conceptelor care reprezintă forme *fizice / grafice*, *fapte*, *evenimente*, *procese*, *stări* etc. Un *obiect* este caracterizat în mod unic prin *identificare*, *comportament* (caracteristică dinamică) și *stare* (caracteristică statică). O clasă este o familie de obiecte cu structură și comportamente identice.

Conform ***OMG*** (***Object Management Group***) o clasă este o implementare care poate fi instanțiată în vederea generării de obiecte cu același comportament.

Comportamentul unui obiect corespunde cu *setul de mesaje* care se transmit obiectului, iar pentru fiecare mesaj există o *metodă* care are acces la starea obiectului. Metodele sunt de mai multe tipuri: *constructori*, *destructori*, *selectori* și *modificatori*.

În esență, rezolvarea unei probleme se va exprima printr-o codificare a *universului problemei* și a raționamentelor pentru procesul demonstrativ. Programul elaborat pentru rezolvarea problemei va fi conceput din două părți:

- date (de intrare și / sau ieșire);
- operații (acestea vor acționa asupra datelor).

Un limbaj de programare modern trebuie să ofere posibilitatea de a construi *tipuri de date* diferite de cele predefinite / standard. *Clasele* reprezintă implementarea unor tipuri abstracte de date (limbajul C++). O ***clasă*** reunește datele și operațiile în cadrul aceluiași tip de date:

Clasă = Structuri de date + Operații

În limbajul C++ definirea unei clase constă din două părți:

- ***declarații*** - lista elementelor componente ale clasei (date membre și metode);
- ***implementarea*** - implementarea metodelor clasei.

O ***structură de date*** reprezintă o structură destinată să stocheze *datele* precum și *metodele* pentru a *crea*, *modifica* și *accesa* datele [12]. Practica *elaborării programelor* pentru aplicațiile simple sau complexe a determinat *utilizarea abstractizării datelor* care constă în a separa algoritmii (procesele de calcul) de structurile de date. În acest fel, rezolvarea problemelor trebuie gândită în *termenii tipurilor abstracte de date* care vor fi folosite în elaborarea aplicațiilor.

Un ***tip abstract de date*** specifică ceea ce se poate face cu o structură (ce operații suportă). O structură de date are trei componente:

- ***tip abstract de date*** (*TAD*) - mulțime de operații care manipulează obiecte abstracte;
- ***structură de stocare*** - suport de memorare a obiectelor abstracte;
- ***implementarea operațiilor*** - definirea operațiilor *TAD* folosind structura de stocare.

Conceptul de algoritm

Practica *dezvoltării aplicațiilor software* (care necesită rezolvarea diverselor tipuri de probleme) a scos la iveală următoarele etape importante [12] (*gândirea obiectuală*: primele două etape ↔ *analiză-proiectare*; *gândirea algoritmică*: ultimele patru etape ↔ *programare-execuție*):

- ***specificarea problemelor*** - descrierea clară și precisă a problemelor indiferent din ce domeniu provin acestea;
- ***proiectarea soluțiilor*** - includerea problemelor în clasa corespunzătoare și alegerea modului de reprezentare a problemelor prin formularea *etapelor* și *procedeelor* corespunzătoare pentru procesele de rezolvare;
- ***implementarea soluțiilor*** - *elaborarea algoritmilor* și *codificarea* acestora într-un limbaj de programare modern;
- ***analiza soluțiilor*** - *eficiența soluțiilor* raportată la resursele utilizate: *memorie*, *timp*, *utilizarea dispozitivelor I/O* etc.;
- ***testarea și depanarea*** - verificarea execuției programului cu diverse seturi de date de intrare pentru a putea răspunde rezolvării oricărei probleme pentru care a fost elaborată aplicația;

- *actualizarea și întreținerea* - adaptarea soluțiilor implementate pentru *eliminarea erorilor* în rezolvarea unei anumite probleme și *compatibilitatea* cu sistemul de calcul și sistemul de operare folosite.

Conform acestor faze iese în evidență interdependența între următoarele activități importante: *Reprezentare* ↔ *Elaborare / Proiectare* ↔ *Execuție*.

Această interdependență este sintetizată prin schema prezentată în figura 1.

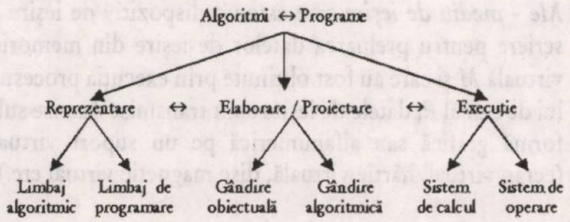


Figura 1

De asemenea, interdependența precizată mai sus se transmite și între componentele de pe nivelul inferior din schema arborescentă alaturată.

Competența* și *experiența în rezolvarea problemelor se pot obține doar dacă permanent se are în vedere această interdependență și dacă se întreprind eforturi pentru însușirea de noi cunoștințe și pentru cunoașterea corespunzătoare a tuturor aspectelor privind ***modelul fizic***, respectiv ***modelul virtual***, aspecte determinate de interdependența ***Sistem de calcul ↔ Algoritmică ↔ Programare.*** Aceasta este ilustrată în figura 2.

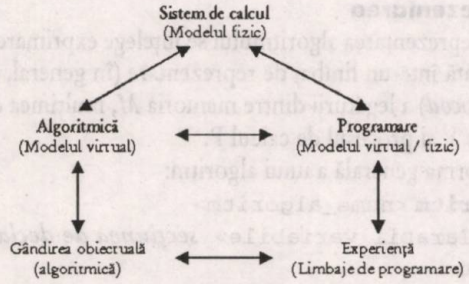


Figura 2

Practica și experiența elaborării programelor pentru rezolvarea problemelor scot în evidență următoarele aspecte foarte importante:

- ***modelul fizic;***
- ***modelul virtual;***
- ***modelul program.***

Modelul fizic

Acest model este dat de *sistemul de calcul*. Modelul trebuie luat în considerare atunci când se proiectează și se elaborează o aplicație; acest aspect reclamă competență în domeniul sistemelor de calcul și perfecționare continuă pentru cel care proiectează și elaborează aplicația.

Modelul virtual

Acest model este dat de *gândirea obiectuală și algoritmică*, *de modul de reprezentare a algoritmilor* și *de mașina virtuală pe care trebuie să se execute algoritmul elaborat*; în timp, acest model a suferit schimbări majore deoarece a fost influențat în continuu de *modelul fizic* și de clasa problemelor care urmau să fie rezolvate.

Modelul program

Acest model este reprezentat *de o îmbinare între modelul fizic* (sistemul de calcul) *și modelul virtual* (***algoritmul***); întotdeauna un program se elaborează într-un limbaj de programare care trebuie să respecte *restricțiile modelului fizic* și *restricțiile modelului virtual.*

Conform acestor aspecte trebuie să existe o echivalență între *algoritm* și *program* indiferent de restricțiile particulare ale modelelor. În esență, un ***program*** reprezintă codificarea într-un limbaj de programare a unui ***algoritm***, iar un ***algoritm*** reprezintă codificarea într-un limbaj de reprezentare a unui ***raționament.***

Programul se va executa pe o mașină fizică / reală reprezentată de un sistem de calcul (SC), iar *algoritmul* va trebui să simuleze execuția pe o mașină virtuală asemănătoare mașinii reale. Acest lucru înseamnă că restricțiile modelului fizic trebuie considerate și în modelul virtual, prin urmare execuția algoritmului trebuie să simuleze execuția programului pe mașina reală.

Comentarii metodologice și pedagogice

- Conform celor de mai sus *este inadecvată utilizarea schemelor logice* în reprezentarea algoritmilor; autorii de manuale de informatică fac o mare greșeală dacă nu țin seama de acest lucru; inventarea și apariția *structurilor de control* în programare au eliminat teoretic schemele logice;
- Utilizarea unui *pseudocod* în reprezentarea algoritmilor care se exprimă cu cuvinte-cheie din limba română nu este oportună dacă se ține seama că algoritmul va fi codificat într-un limbaj de programare modern: *C, C++, Java, Pascal, Foxpro, Oracle, Prolog* etc.; încă mai există manuale care utilizează această formă de reprezentare; pe viitor astfel de manuale nu trebuie să mai existe;
- ***Activitățile de laborator sunt esențiale în procesul de învățare***; profesorii de informatică sunt obligați să îmbine cât mai eficient *orele de predare cu orele de laborator* prin care de fapt se încheie ciclul pentru rezolvarea unei probleme cu calculatorul: *problema → modelul matematic → algoritmul → programul → calculator → rezultate → verificare soluții.*

Se observă echivalența *algoritm ↔ program ↔ SC* modulo operațiile realizate de algoritm în spațiul virtual, respectiv operațiile realizate de *SC* prin execuția programului corespunzător algoritmului. Algoritmul va trebui să simuleze toate operațiile declanșate prin lansarea în execuție a programului corespunzător algoritmului.

Definiția conceptului de algoritm

Un *algoritm* este sistemul virtual $A = (M, V, P, R, Di, De, Mi, Me)$ caracterizat de următoarele aspecte:

- *elaborare*;
- *reprezentare*;
- *execuție*;
- *corectitudine și analiză.*

Algoritmul

Sistemul A este constituit din următoarele elemente:

- *M - memorie virtuală (internă)* utilizată pentru stocarea temporară a informațiilor destinate variabilelor din mulțimea de variabile V; asupra variabilelor acționează procesul de calcul P care are acces la memorie prin rezervarea de memorie pentru variabilele din mulțimea V; inițial memoria virtuală este folosită pentru rezervare de memorie pentru variabilele din mulțimea V, după care este utilizată pentru scrierea și citirea de informații în locațiile corespunzătoare variabilelor utilizate în procesul de calcul P;
- *V - mulțime de variabile / structuri de date* definite conform raționamentului R corespunzător rezolvării unei probleme și care utilizează memoria M prin locații de memorie pentru fiecare tip de variabilă din V; locațiile de memorie rezervate variabilelor din V sunt utilizate de procesul de calcul P care prin execuția instrucțiunilor care constituie P, schimbă valorile (starea) locațiilor de memorie corespunzătoare variabilelor în conformitate cu implementarea raționamentului R;
- *P - proces de calcul* care este reprezentat de o colecție de instrucțiuni / comenzi exprimate într-un limbaj de reprezentare (cel mai utilizat fiind *pseudocodul*); folosind memoria virtuală M și mulțimea de variabile V, colecția de instrucțiuni implementează / codifică tehnicile și metodele care constituie raționamentul R conceput special pentru rezolvarea unei clase de probleme; execuția instrucțiunilor din P determină o dinamică a valorilor locațiilor de memorie corespunzătoare din V; după execuția tuturor instrucțiunilor din P, soluția / soluțiile problemei se află în anumite locații de memorie care constituie datele de ieșire De;
- *R - raționament de rezolvare* exprimat prin diverse tehnici și metode specifice domeniului din care face parte clasa de probleme supuse rezolvării (matematică, fizică, chimie etc.), care îmbinate cu tehnici de programare corespunzătoare realizează acțiuni / procese logice prin utilizarea memoriei virtuale M și a mulțimii de variabile V;
- *Di - date de intrare* care reprezintă valori ale unor parametri care caracterizează ipoteze de lucru / stări inițiale; valorile datelor de intrare sunt stocate în memoria M prin intermediul instrucțiunilor de citire / intrare care utilizează mediul de intrare Mi; acesta este un dispozitiv virtual pentru citirea datelor de intrare;
- *De - date de ieșire* care reprezintă valori ale unor parametri care caracterizează soluția / soluțiile problemei invocate de cerințele problemei / stările finale; valorile datelor de ieșire sunt obținute din valorile intermediare ale unor variabile generate de execuția instrucțiunilor din procesul de calcul P și care în final sunt stocate în memoria M în vederea transmiterii / scrierii lor către mediul de ieșire Me; acesta este un dispozitiv virtual pentru reprezentarea / scrierea datelor de ieșire sub forma grafică sau alfanumerică;
- *Mi - mediu de intrare* care este un dispozitiv virtual de intrare / citire pentru citiri virtuale ale valorilor datelor de intrare pentru a fi stocate în memoria virtuală M;
- *Me - mediu de ieșire* care este un dispozitiv de ieșire / scriere pentru preluarea datelor de ieșire din memoria virtuală M și care au fost obținute prin execuția procesului de calcul P; datele de ieșire sunt transmise / scrise sub formă grafică sau alfanumerică pe un suport virtual (ecran virtual, hârtie virtuală, disc magnetic virtual etc.).

Elaborarea

Elaborarea / conceperea algoritmului înseamnă construirea unui proces demonstrativ / computațional care va constitui raționamentul de rezolvare R și care va îngloba metode și tehnici eficiente pentru găsirea soluției / soluțiilor problemelor pentru care se elaborează algoritmul; elaborarea raționamentului de rezolvare R constă din următoarele:

- definirea *datelor de intrare* (Di) și a *datelor de ieșire* (De);
- aplicarea metodelor și tehnicilor utilizate pentru rezolvare;
- definirea mulțimii variabilelor V utilizate în rezolvare;
- codificarea raționamentului de rezolvare R într-un limbaj de reprezentare (de tip *pseudocod*) va constitui procesul de calcul P.

Reprezentarea

Prin reprezentarea algoritmului se înțelege exprimarea formalizată într-un limbaj de reprezentare (în general, de tip *pseudocod*) a legăturii dintre memoria M, mulțimea de variabile V și procesul de calcul P.

Forma generală a unui algoritm:

```
algoritm <nume_algoritm>
<declaratii_variabile>  secțiunea de declarații
start
   <procesul_de_calcul_P>  corpul algoritmului
stop
```

Execuția

Algoritmii se consideră executați pe *mașini abstracte / virtuale* (ale căror caracteristici le "abstractizează" pe cele ale mașinilor de calcul / sistemelor de calcul existente la un moment dat).

Astfel de modele de mașini de calcul sunt:

- *mașina Turing sau diverse automate cu stivă* (1956-1972, vezi A. Aho, J. Hopcroft, J.D. Ullman, *Data strutures and algorithms*, Addison Wesley publishing Co.,1983);
- *mașina MIX* (D. Knuth, *Knuth's MIX Language*, 1973);
- *mașina RAM* (A. Aho, J. Hopcroft, J.D. Ullman, *The Random Access Machine*, 1974);

- **mașina ASM** (*Abstract State Machines - A Formal Method for Specification and Verification*);
- **mașina VDM** (Bjorner, Jones, *Vienna Development Method*, *mașină abstractă Prolog*, 1980);
- **mașina WAM**(*Warren Abstract Machine*, *mașină abstractă Prolog*, 1983);
- **mașina pseudocod** (T. Cormen, L. Leiserson, R. Rivest, *Pseudo-Code Language*, 1994).

Prin intermediul unei *mașini abstracte / virtuale* (care simulează o mașina reală / sistem de calcul) execuția algoritmului înseamnă interpretarea reprezentării algoritmului ca un mecanism virtual de tip dinamic care are o memorie virtuală *M*, un proces de calcul *P*, mediu de intrare *Mi*, mediu de ieșire *Me* și toate aceste componente se vor confunda cu elementele corespunzătoare mașinii virtuale, facând excepție procesul de calcul *P* care este "*lansat în execuție*" de unitatea centrală a mașinii virtuale.

Prin "*lansarea în execuție*" a procesului de calcul *P* de către mașina virtuală se înțelege exercitarea tuturor funcțiilor unității centrale (având memorie și procesor) pentru a simula toate procesările invocate de instrucțiunile procesului de calcul *P*.

Execuția algoritmului va însemna generarea de procese ce se vor desfășura în timp și care vor folosi memorie, dispozitive de calcul și dispozitive de intrare / ieșire la fel cum se întâmplă cu lansarea în execuție a formei executabile a unui program pe un sistem de calcul / calculator real.

În felul acesta, algoritmul, care este sistemul virtual *A* = (*M*, *V*, *P*, *R*, *Di*, *De*, *Mi*, *Me*), poate fi considerat ca un "*sistem de calcul*" *virtual* având componentele de bază: *M* (memorie internă), *P* (procesor), *Mi* (mediu de intrare) și *Me* (mediu de ieșire).

Ținând seama că un program este codificarea algoritmului într-un limbaj de programare, definiția unui program este asemănătoare cu definiția unui algoritm cu deosebirea că nu mai este necesară prezența raționamentului *R*, iar reprezentarea programului se va face conform sintaxei și semanticii limbajului de programare ales; pentru ambele concepte "*execuția*" evidențiază structura unui *sistem cibernetic* care se află la baza arhitecturii unui sistem de calcul (ansamblu de componente *hardware* (dispozitive) și *software* (programe) care oferă servicii utilizatorului pentru execuția programelor care implementează rezolvarea unor probleme).

Figura 3

Mașina abstractă / virtuală va funcționa după modelul unei mașini de calcul reale.

Schema de funcționare și fluxul informațional sunt ilustrate în figura 4.

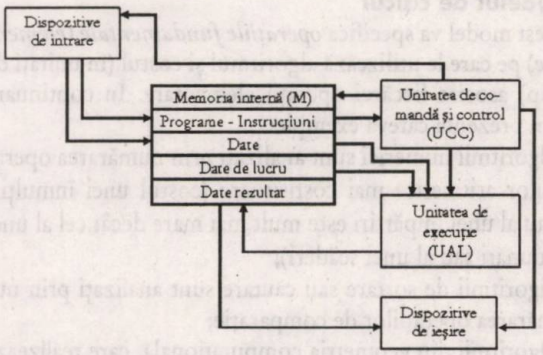


Figura 4

În concluzie, învățarea algoritmilor trebuie să fie precedată de învățarea elementelor de bază despre sistemele de calcul și să preceadă învățarea programării, și anume: *sisteme de calcul* → *algoritmi* → *programare*.

Corectitudinea și analiza

Corectitudinea algoritmului este exprimată de ***corectitudinea parțială*** (procesul de calcul se termină - timpul de execuție este finit - pentru orice date de intrare dintr-un anumit domeniu de valori) și ***corectitudinea totală*** (pentru orice date de intrare dintr-un domeniu de valori, procesul de calcul determină valori corecte conform scopului / funcției algoritmului).

Există diverse metode pentru verificarea celor două componente ale corectitudinii (de exemplu, pentru *algoritmul lui Euclid* reprezentat corect în pseudocod, ***corectitudinea parțială*** este verificată de faptul că șirul valorilor resturilor obținute din împărțirile succesive este un șir convergent către 0, iar ***corectitudinea totală*** este verificată de *metoda lui Euclid* printr-o simplă demonstrație matematică).

Elaborarea aplicațiilor informatice necesită testarea programelor care implementează diverși algoritmi pentru ca programatorul să se convingă de corectitudinea algoritmilor concepuți.

Analiza algoritmului se referă la *spațiul de memorie* utilizat și la *timpul necesar executării algoritmului*; această analiză constă în măsurarea și descrierea (cantitativă) a *performanțelor* algoritmului. Astfel, este posibilă compararea diverselor soluții algoritmice pentru aceeași problemă.

De regulă, resursa ***timp*** este mai critică (importantă) decât resursa ***spațiu de memorie***. Cu toate acestea, există numeroase situații în activitatea de proiectare și implementare a unor noi algoritmi, când se stabilește un compromis între cerințele de spațiu de memorie și cele referitoare la timpul de execuție.

Analiza unui algoritm cuprinde următoarele abordări:
- stabilirea unui *model de calcul*;
- stabilirea *timpului de calcul / execuție*;
- stabilirea *ratei de creștere*.

Modelul de calcul

Acest model va specifica *operațiile fundamentale* (*elementare*) pe care le utilizează algoritmul și costul (în unități de timp) asociat fiecărei operații elementare. În continuare vom prezenta câteva exemple:
- algoritmii numerici sunt analizați prin numărarea operațiilor aritmetice mai costisitoare (costul unei înmulțiri sau al unei împărțiri este mult mai mare decât cel al unei adunări sau al unei scăderi);
- algoritmii de sortare sau căutare sunt analizați prin numărarea operațiilor de comparație;
- algoritmii din geometria computațională, care realizează operații asupra poligoanelor, sunt analizați prin numărarea vârfurilor sau muchiilor prelucrate.

Timpul de execuție

Există așa-numitele *măsuri de complexitate* care descriu aspectul de performanță care trebuie măsurat: timpul de execuție în *cazul cel mai defavorabil*, în *cazul mediu* și în *cazul amortizat* (marginea superioară în cazul cel mai defavorabil).

Aceste măsuri de complexitate depind de volumul datelor de intrare.

Rata de creștere

Prin analiza asimptotică se stabilește rata de creștere a timpului de execuție în funcție de volumul setului de date de intrare.

Analiza asimptotică exprimă creșterea timpului de execuție al unui algoritm în cazul creșterii (spre infinit) a volumului setului de date de intrare.

Timpul de execuție este exprimat de funcția $f(n)$, n fiind numărul de elemente de intrare, iar rata de creștere a funcției $f(n)$ este dată de o funcție notată prin $T(n)$.

De exemplu, dacă $f(n) = a \cdot n^2 + b \cdot n + c$, unde a, b și c sunt constante, atunci când n crește spre infinit, termenii de ordin 1 și 0 sunt nesemnificativi, și astfel, rata de creștere a lui $f(n)$ este funcția $T(n) = n^2$.

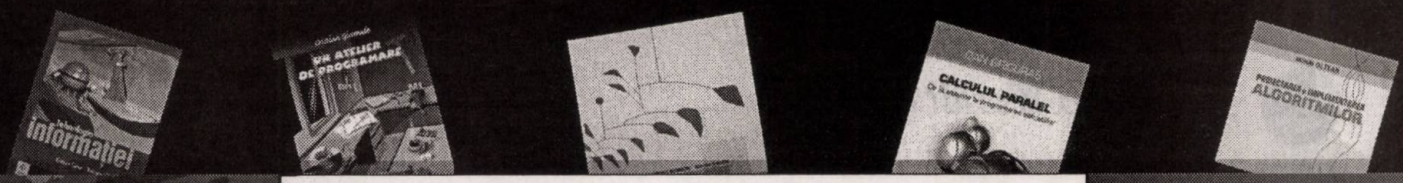
În funcție de variația funcției $T(n)$ există următoarele categorii de algoritmi: *liniari*, *pătratici*, *cubici*, *exponențiali*, *logaritmici*, *liniar-logaritmici* etc.

Notațiile utilizate în analiza asimptotică a algoritmilor sunt următoarele:
- $O(f(n))$ care reprezintă clasa funcțiilor care cresc mai puțin decât $f(n)$ când n tinde la infinit;
- $o(f(n))$ care reprezintă clasa funcțiilor care cresc strict mai lent decât $f(n)$ când n tinde la infinit;
- $\Omega(f(n))$ care reprezintă clasa funcțiilor care nu cresc mai încet decât $f(n)$ când n tinde la infinit;
- $\Theta(f(n))$ care reprezintă clasa funcțiilor care cresc cu aceeași rată ca și $f(n)$ când n tinde la infinit.

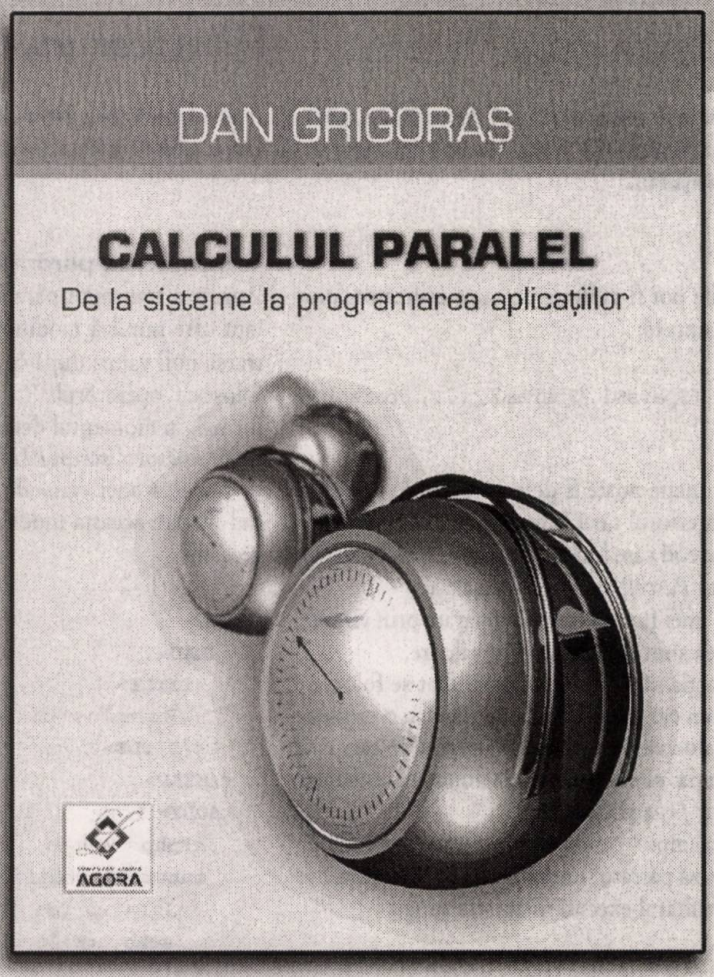
Bibliografie

1. Albeanu Gr., *Algoritmi și limbaje de programare*, Universitatea "Spiru Haret", Editura România de Mâine, București, 2000
2. Apostol C., Roșca I. Gh., Roșca V., Ghilic-Micu B., *Introducere în programare. Teorie și aplicații*, Editura București, 1993
3. Georgescu H., *Programare concurentă. Teorie și aplicații*, Editura Tehnică, București, 1996
4. Kinston J. H., *Algorithms and Structures Design. Correctness, Analysis*, Addison Wesley, Longman, 1998
5. Mitrana V., *Provocarea algoritmilor. Probleme pentru concursurile de informatică*, Editura Agni, București, 1994
6. Odăgescu I., *Optimizarea algoritmilor*, Editura Militară, București, 1991
7. Perjeriu E., Văduva I., *Îndrumar pentru lucrări de laborator la cursul de Bazele Informaticii*, Universitatea din București, 1986
8. Popovici C., Georgescu H., State L., *Bazele Informaticii*, vol. I, Universitatea din București, 1990
9. Skiena S., *The Algorithm Design Manual*, Springer Verlag, New York, Inc., 1998
10. Vlada M., *Informatica*, Universitatea din București, Editura Ars Docendi, București, 1999
11. Vlada M., *Poligoane stelate. Problema lui Hopf și Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348
12. Zaharia M. D., *Structuri de date și algoritmi. Exemple în limbajele C și C++*, Editura Albastră, Cluj-Napoca, 2002
13. Cristea V., Giumale C., Kalisz E., Păunoiu Al., *Limbajul C standard*, Editura Teora, București, 1992
14. Popovici M. D., Popovici M. I., *C++. Tehnologia orientată spre obiecte. Aplicații*, Editura Teora, București, 2000
15. www.math.gatech.edu/~thomas/FC/fourcolor.html
16. mathworld.wolfram.com/Four-ColorProblem.html
17. spicerack.sr.unh.edu/~student/tutorial/fourColor/FourColor.html
18. mathcentral.uregina.ca/RR/database/RR.09.97/fisher1.html
19. www.uccs.edu/~asoifer/book5.html
20. www.uccs.edu/~asoifer/solve98.html
21. www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg
22. mathworld.wolfram.com/Algorithm.html
23. www.cs.rit.edu/~atk/Java/Sorting/sorting.html
24. faculty.juniata.edu/rhodes/cs2/ch129.htm
25. www.scism.sbk.ac.uk/law/Section5/chap1/ s5c1p2.htm
26. www.ics.uci.edu/~eppstein/161/people.html
27. www.cs.pitt.edu/~kirk/algorithmcourses/
28. www.gnacademy.org/text/cc/
29. java.sun.com/docs/books/tutorial/java/concepts/
30. www.accu.org/acornsig/public/articles/oop_c.html
31. loki.cs.brown.edu:8081/webae/full.html

Domnul conf. dr. Marin Vlada este cadru didactic la Universitatea București și poate fi contactat prin e-mail la adresa vlada@chem.unibuc.ro.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS
AGORA

Hello World! (19)

Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

Acest articol este dedicat funcțiilor, claselor și obiectelor PHP. Vor fi prezentate detalii referitoare la modul în care pot fi utilizate aceste elemente ale limbajului.

Funcții

În *PHP* 4 funcțiile pot fi definite de către utilizator folosind următoarea sintaxă:

```
function numef ($param1, $param2, ..., $paramN){
  //instrucțiuni
}
```

În *PHP* 4 o funcție poate fi definită oriunde în cadrul *script*-ului și în interiorul unei funcții poate să apară orice secvență validă de cod care include definirea de alte funcții (similar limbajului *Pascal*) și definiții de clase.

Argumentele unei funcții trebuie separate prin virgulă, și, implicit, acestea sunt transmise prin valoare.

Pentru ca funcția să returneze un rezultat se folosește construcția **return** care primește ca parametru o expresie care reprezintă valoarea funcției. În momentul în care este întâlnită construcția **return**, execuția funcției se încheie (similar limbajelor *Java* și *C/C++*).

Următorul exemplu reprezintă o funcție simplă în *PHP* care calculează pătratul unui număr, iar în figura 1 se poate observa rezultatul executării acestui script:

```
<HTML>
  <HEAD>
    <TITLE>Exemplu funcție simplă</TITLE>
  </HEAD>
  <BODY>
    <?php
      function patrat($n){
        return $n*$n;
      }
      echo "4^2=<B>".patrat(4)."</B>";
    ?>
  </BODY>
</HTML>
```

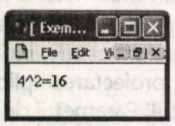


Figura 1

Din exemplul anterior rezultă că o funcție în *PHP* 4 poate primi ca parametru valori constante ca și în cazul majorității limbajelor de programare moderne.

Transmiterea parametrilor prin referință

Pentru a transmite parametri unei funcții prin referință, fapt care implică modificarea valorii parametrilor și păstrarea noii valori după ce execuția funcției s-a încheiat, se folosește operatorul "&" înaintea numelui parametrului formal, în momentul definirii funcției.

Următorul script *PHP* indică modul în care se modifică valoarea unei variabile în interiorul unei funcții și modul în care această modificare este resimțită în exteriorul acesteia:

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu modificare parametri
    </TITLE>
  </HEAD>
  <BODY>
    <?php
    function modificInt($s){
      $s.=" prima funcție.";
      echo "<B>În modificInt:</B> ".$s."<BR>";
    }
    function modificExt(&$s){
      $s.=" a doua funcție.";
      echo "<B>În modificExt:</B> ".$s."<BR>";
    }
    $s="Ieșire din";
    echo "<B>În script:</B> ".$s."<BR>";
    modificInt($s);
    echo "<B>În script:</B> ".$s."<BR>";
    modificExt($s);
    echo "<B>În
          script:</B"
          .$s."<BR>";
    ?>
  </BODY>
</HTML>
```

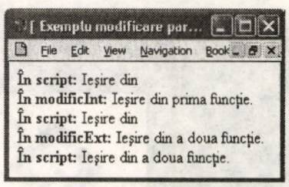


Figura 2

În figura 2 se poate observa efectul executării acestui *script PHP* 4.

Nu există posibilitatea de supraîncărcare a unei funcții, de redefinire a ei după ce aceasta a fost definită în cadrul scriptului respectiv și nu există nici posibilitatea de anulare a unei funcții. O funcție poate fi apelată înainte de definirea ei.

Parametri cu valori implicite

În *PHP* parametrii formali pot avea valori implicite, și, în cazul în care parametrul actual lipsește, atunci se va considera că are valoarea implicită.

Următorul exemplu ilustrează modul de folosire al funcțiilor când acestea au parametri formali cu valori implicite:

```
<HTML>
  <HEAD>
    <TITLE>Exemplu valori implicite</TITLE>
</HEAD>
  <BODY>
    <?php
      function comanda($s="cafea"){
        return "Ați comandat ".$s.".";
      }
      echo comanda();
      echo "<BR>";
      echo comanda("suc");
    ?>
  </BODY>
</HTML>
```

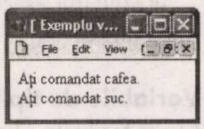


Figura 3

În figura 3 se poate observa rezultatul executării acestui script *PHP*.

În cazul în care se folosesc parametri cu valori implicite este necesar ca orice parametru care are o valoare implicită să se afle în partea dreaptă a tuturor parametrilor pentru care nu se folosesc valori implicite, în caz contrar interpretorul *PHP* nu poate să decidă cărui parametru să-i atribuie valoarea de pe o anumită poziție din lista cu parametri.

De exemplu, dacă avem o funcție a cărei declarație este `function transform($baza=10, $nr)` și care returnează rezultatul transformării lui `$nr` din baza 16 în baza `$baza`, a cărei valoare implicită este 10, dacă se apelează `transform(50)`, interpretorul nu atribuie valoarea 50 parametrului `$nr`, ci parametrului `$baza` și generează o eroare deoarece lipsește valoarea parametrului `$nr`.

Funcții cu număr variabil de parametri

O altă facilitate importantă a limbajului *PHP* este aceea că oferă programatorului posibilitatea de a utiliza funcții care au un număr nedeterminat de parametri. Funcțiile care folosesc un număr variabil de parametri nu au nici o particularitate în ceea ce privește definirea lor. Aceste funcții se definesc la fel ca cele prezentate anterior, dar pentru a putea accesa parametri se vor folosi următoarele funcții predefinite:

- `func_num_args()` - această funcție returnează numărul parametrilor funcției care a apelat-o; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;
- `func_get_arg(arg_num)` - returnează valoarea parametrului care se află pe poziția `arg_num` în lista de parametri; primul parametru are numărul de ordine 0; dacă este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;
- `func_get_args()` - returnează un tablou unidimensional care conține valorile parametrilor pe care funcția apelantă i-a primit; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare.

În continuare vom prezenta exemple de utilizare a acestor funcții.

Primul exemplu afișează lista parametrilor funcției folosind funcțiile `func_num_args` și `func_get_arg`, iar al doilea exemplu afișează aceeași listă folosind numai funcția `func_get_args`.

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu parametri multipli
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function lista_parametri(){
        for($i=0;$i<func_num_args();$i++){
          print_r(func_get_arg($i));
          echo "<BR>";
        }
      }
      echo lista_parametri("Comanda:",1,
        "calculator", 2, "procesoare",
        "configurație", array("local",2,3));
    ?>
  </BODY>
</HTML>
```

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu parametri multipli
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function lista_parametri(){
        foreach(func_get_args() as $i){
          print_r($i);
```

```
            echo "<BR>";
        }
    }
    echo lista_parametri("Comanda:",1,
      "calculator", 2, "procesoare",
      "configuraţie", array("local",2,3));
  ?>
 </BODY>
</HTML>
```

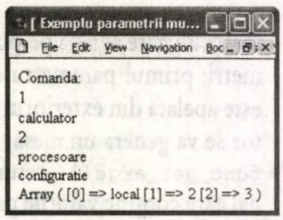


Figura 4

În figura alăturată se poate observa rezultatul executării celor două scripturi *PHP*.

Valorile returnate de funcţii

Rezultatul obţinut după apelarea unei funcţii poate avea orice tip. O funcţie poate să returneze chiar şi liste sau obiecte.

În *PHP* există un caz special de rezultat numit referinţă. Pentru ca o funcţie să poată returna o referinţă, aceasta trebuie declarată folosindu-se operatorul '&' înaintea numelui funcţiei. Acest operator trebuie să apară înaintea numelui funcţiei şi în mometul când o variabilă primeşte ca valoare referinţa rezultată din apelul funcţiei.

În exemplul următor se defineşte o funcţie al cărui rezultat îl constituie o referinţă:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu rezultat de tip referinţă
  </TITLE>
 </HEAD>
 <BODY>
  <?php
   function &refer(){
    global $s;
    return $s;
   }
   $s = "Acesta este conţinutul variabilei
     referite cu ajutorul funcţiei.";
   $z = &refer();
   echo $z;
  ?>
 </BODY>
</HTML>
```

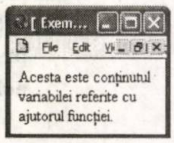


Figura 5

În figura 5 se poate observa rezultatul execuţiei acestui script *PHP*.

Spre deosebire de majoritatea limbajelor de programare moderne, o funcţie *PHP* poate să returneze o referinţă la o variabilă care a fost declarată în interiorul funcţiei, însă acest lucru nu este indicat deoarece, în anumite cazuri, poate duce la efecte neaşteptate ale executării unui script *PHP*. În alte limbaje de programare efectele devin uneori fatale.

Următorul script *PHP* arată unul dintre cazurile în care variabila a cărei referinţă este returnată este declarată în interiorul funcţiei:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu #2 rezultat de tip referinţă
  </TITLE>
 </HEAD>
 <BODY>
  <?php
   function &refer(){
    $s = "Acesta este conţinutul variabilei
        referite cu ajutorul funcţiei.";
    return $s;
   }

   $z = &refer();
   echo $z;
  ?>
 </BODY>
</HTML>
```

Rezultatul executării acestui script este acelaşi cu cel din figura 5.

Variabile de tip funcţie

O altă facilitate a limbajului *PHP* în ceea ce priveşte funcţiile este aceea că suportă variabilele de tip funcţie. Acest lucru este util atunci când se folosesc liste de funcţii pentru prelucrarea anumitor tipuri de date. Pentru a atribui un nume de funcţie unei variabile în *PHP* se foloseşte acelaşi construcţie ca în cazul atribuirii unui şir de caractere, şi anume, o variabilă va primi ca valoare numele funcţiei scris între ghilimele simple sau duble. În cazul în care interpretorul *PHP* găseşte un nume de variabilă urmată de o listă de parametri, acesta caută funcţia pe care variabila o referă şi în cazul în care există, o execută. Variabilele de tip funcţie nu funcţionează cu construcţii ale limbajului ca **echo**, **unset**, **isset**, **empty**, **include** etc.

Următorul script *PHP* ilustrează modul de lucru cu variabile de tip funcţie, iar în figura 6 se poate observa rezultatul executării acestuia:

```
<HTML>
 <HEAD>
  <TITLE>
   Variabile de tip funcţie
  </TITLE>
 </HEAD>
 <BODY>
  <?php
   function produs($a, $b){
    return $a*$b;
   }
```

```
    function suma($a, $b){
      return $a+$b;
    }

    $operatie = 'produs';
    $rez = $operatie(4, 5);
    echo "4 * 5 = <B>".$rez."</B><BR>";
    $operatie = 'suma';
    $rez = $operatie(4, 5);
    echo "4 + 5 = <B>".$rez."</B><BR>";
  ?>
 </BODY>
</HTML>
```

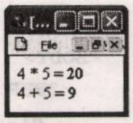


Figura 6

Clase și obiecte

În continuare vom presupune că paradigmele programării orientate pe obiecte sunt cunoscute, așadar nu vom prezenta aspectele teoretice referitoare la clase și obiecte, ci doar vom descrie modul în care acestea pot fi utilizate în *PHP*. Pe scurt, o clasă este o colecție de variabile și funcții care operează asupra variabilelor respective. Sintaxa folosită pentru declararea unei clase în *PHP*:

```
class nume_clasă{
  // date membre
  var nume_variabilă_1
  ...
  var nume_variabilă_m
  // metode
  function nume_funcție_1(parametri){
    ... // definiția funcției
  }
  ...
  function nume_funcție_n(parametri){
    ... // definiția funcției
  }
}
```

Pentru numele unei clase poate fi utilizat orice identificator permis în *PHP* cu o singură excepție: `stdclass`; acest identificator este folosit de *PHP* în scopuri interne.

În *PHP* funcțiile ale căror identificatori încep cu '__' sunt considerate funcții magice și utilizarea acestora nu este recomandată.

În *PHP*, datele membre nu pot fi inițializate decât cu valori constante. Pentru a inițializa variabilele cu valori care nu sunt constante trebuie folosit un constructor.

Așadar, nici una dintre următoarele inițializări nu este corectă:

```
class Nepermis{
  var $data = date("Y-m-d");
  var $nume = $prenume;
  var $dest = 'Mihai ' . 'Claudiu';
  var $obiecte = array("minge", "pantof");
}
```

Obiecte

În *PHP* clasele sunt considerate a fi tipuri de date; ele pot fi privite ca fiind "amprentele" variabilelor propriu-zise. Pentru a crea o variabilă al cărei tip este o clasă trebuie utilizat operatorul **new**.

În continuare vom defini o clasă `Aritmetica` cu două date membre `x` și `y` care sunt numere întregi și două metode care realizează adunarea, respectiv înmulțirea lor.

```
class Aritmetica{
  var x = 2;
  var y = 3;

  function Suma(){
    return $this->x + $this->y;
  }
  function Produs(){
    return $this->x * $this->y;
  }
}
```

Pentru a crea un obiect de tipul `Aritmetica` vom utiliza o instrucțiune de tipul:

```
$aritm = new Aritmetica;.
```

Acum putem utiliza metodele clasei; pentru a afișa suma sau produsul celor două numere vom putea apela cele două metode astfel:

```
echo $aritm->Suma();
echo $aritm->Produs();
```

Vom obține rezultatele `5`, respectiv `6`. Valorile datelor membre pot fi și ele modificate prin instrucțiuni de tipul:

```
$aritm->x = 5;
$aritm->y = 4;
```

Dacă, în urma modificării apelăm din nou metodele `Suma()` și `Produs()`, rezultatele vor fi `9`, respectiv `20`.

Vom prezenta în continuare un document *PHP* integral în care se definește o clasă `HelloWorld` care are ca date membre un șir de caractere care conține mesajul `Hello World!` și o metodă care returnează acest mesaj. Efectul *script*-ului este ilustrat în figura 7.

```
<HTML>
 <HEAD>
  <TITLE>Hello World!</TITLE>
 </HEAD>
 <BODY>
  <?PHP
   class HelloWorld{
    var $hello = "Hello World!";
    function Hello(){
     return $this->hello;
    }
   }
```

Figura 7

```
    $helloWorld = new HelloWorld;
    echo $helloWorld->hello()."<BR>";
  ?>
 </BODY>
</HTML>
```

În acest exemplu a fost utilizată pseudo-variabila **$this**. Aceasta este folosită pentru a indica faptul că se operează asupra unei date membre a obiectului curent.

Extinderea claselor

Deseori este necesară definirea unor clase cu proprietăți (date membre) și metode asemănătoare. Pentru a ușura definirea unor astfel de clase a fost introdus conceptul de *extindere* (derivare) a claselor.

O clasă derivată va păstra toate proprietățile și metodele clasei pe care o extinde și poate conține diferite proprietăți și metode noi.

Păstrarea de către clasa extinsă a metodelor și proprietăților clasei din care este derivată poartă denumirea de *moștenire*.

Clasa pe care o extinde o clasă derivată poartă denumirea de ***clasă de bază*** sau ***clasă părinte***. Nu există nici o posibilitate de a elimina din clasa derivată anumite proprietăți sau metode ale clasei de bază.

O anumită clasă poate avea o singură clasă părinte; așadar, în *PHP* nu este permisă moștenirea multiplă.

Pentru a extinde o anumită clasă se utilizează cuvântul cheie **extends**.

În următorul exemplu vom extinde clasa `Aritmetica`; vom adăuga încă o variabilă și vom crea două noi funcții: una pentru calculul sumei celor trei variabile și una pentru calcularea produsului lor.

```
class Aritmetica3 extends Aritmetica{
  var z = 4;

  function Suma3(){
    return $this->x + $this->y + $this->z;
  }
  function Produs3(){
    return $this->x * $this->y * + $this->z;
  }
}
```

Dacă definim un obiect prin intermediul unei instrucțiuni de genul:

```
$aritm3 = new Aritmetica3;
```

atunci pentru acest obiect vom putea utiliza atât metodele definite în cadrul clasei `Aritmetica3`: `Suma3()` și `Produs3()`, cât și metodele definite în cadrul clasei de bază `Aritmetica`: `Suma()` și `Produs()`.

În continuare vom prezenta întregul document *PHP* care ilustrează modul în care pot fi create și utilizate clasele derivate. Efectul execuției acestui script este prezentat în figura 8.

```
<HEAD>
 <BODY>
  <?PHP
    class Aritmetica{
      var $x = 2;
      var $y = 3;
      function Suma(){
        return
     $this->x +
          $this->y;
      }
      function Produs(){
        return $this->x * $this->y;
      }
    }
    class Aritmetica3 extends Aritmetica{
      var $z = 4;
      function Suma3(){
        return $this->x + $this->y +
                                $this->z;
      }
      function Produs3(){
        return $this->x * $this->y *
                                $this->z;
      }
    }

    $aritm3 = new Aritmetica3;
    echo "<B>Înainte de modificare:</B>";
    echo "<BR>";
    echo "Suma primelor două numere: ";
    echo $aritm3->Suma()."<BR>";
    echo "Produsul primelor două numere: ";
    echo $aritm3->Produs()."<BR>";
    echo "Suma celor trei numere: ";
    echo $aritm3->Suma()."<BR>";
    echo "Produsul celor trei numere: ";
    echo $aritm3->Produs()."<BR>";
    $aritm3->x = 5;
    $aritm3->y = 4;
    $aritm3->z = 3;
    echo "<BR><BR>";
    echo "<B>După modificare:</B><BR>";
    echo "<BR>";
    echo "Suma primelor două numere: ";
    echo $aritm3->Suma3()."<BR>";
    echo "Produsul primelor două numere: ";
    echo $aritm3->Produs3()."<BR>";
    echo "Suma celor trei numere: ";
    echo $aritm3->Suma3()."<BR>";
    echo "Produsul celor trei numere: "
    echo $aritm3->Produs3()."<BR>";
  ?>
 </BODY>
</HTML>
```

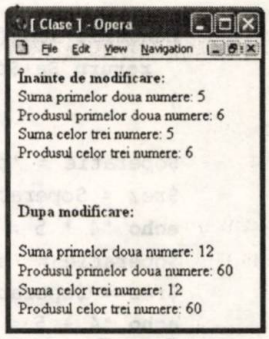


Figura 8

Relația dintre o clasă derivată și clasa pe care o extinde poartă denumirea de ***relație părinte - fiu***. În *PHP* clasele trebuie definite înaintea utilizării lor; așadar clasa părinte va fi definită întotdeauna înaintea clasei fiu.

Constructori

Un constructor este o metodă (funcție) a unei clase care este apelată automat în momentul în care este creată o nouă instanță a clasei (cu ajutorul operatorului **new**).

În *PHP* este considerată ca fiind un constructor orice funcție care are același nume cu clasa în interiorul căreia este definită.

Constructorii pot fi folosiți pentru inițializarea datelor membre cu valori care nu sunt constante. Ei pot avea argumente, iar acestea pot fi opționale. Pentru a putea utiliza clasa fără a specifica nici un parametru în momentul creării unui obiect, se recomandă stabilirea unor valori implicite pentru toate argumentele constructorului.

În cazul în care nu este definit un constructor pentru o anumită clasă, se utilizează constructorul clasei de bază, dacă acesta există.

De exemplu, pentru următoarea secvență de cod, în momentul creării obiectului corespunzător variabilei `$b`, va fi apelat constructorul clasei `A`.

```
class A{
  function A(){
    echo "Constructorul clasei A<BR>";
  }
  function B(){
    echo "O funcție obișnuită a clasei A.<BR>";
  }
}

class B extends A{
  function C(){
    echo "O funcție obișnuită a clasei B.<BR>";
  }
}

$b = new B;
```

În mai multe limbaje de programare constructorul unei clase derivate apelează automat constructorul clasei de bază. În *PHP* apelul constructorului clasei de bază trebuie să fie explicit dacă este necesară executarea operațiilor corespunzătoare.

În majoritatea limbajelor de programare există funcții speciale numite destructori care sunt apelate automat în momentul "distrugerii" unui obiect. În *PHP* nu există destructori.

Operatorul ::

Uneori este utilă folosirea unor metode sau variabile ale clasei de bază sau ale unei clase care nu a fost instanțiată încă. În acest scop a fost introdus operatorul `::`.

Pentru a descrie modul de utilizare al acestui operator vom prezenta mai întâi un exemplu:

```
class A{
  function exemplu(){
    echo "Funcția clasei de bază.<BR>";
  }
}
class B extends A{
  function exemplu(){
    echo "Funcția redefinită<BR>\n";
    A::exemplu();
  }
}
A::exemplu();
$b = new B;
$b->exemplu();
```

Prin intermediul instrucțiunii `A::exemplu();` este apelată metoda `exemplu()` a clasei `A`, așadar se afișează mesajul `Funcția clasei de bază` cu toate că nu există nici un obiect care este o instanță a acestei clase, deci nu putem scrie o instrucțiune de tipul `$a->exemplu();`. În schimb apelăm metoda `$b->exemplu();` ca "o funcție a clasei" și nu ca "o funcție a unui obiect".

Putem avea funcții ale claselor, dar nu putem avea variabile ale claselor. De fapt, în momentul unui astfel de apel nu se creează nici un obiect care este instanță a clasei respective. Ca urmare, o funcție a unei clase nu poate opera asupra unor proprietăți ale clasei, dar poate utiliza variabile locale sau globale. În plus, o astfel de funcție nu poate utiliza pseudo-variabila **`$this`**.

În exemplul anterior, în cadrul clasei `B` este redefinită funcția `exemplu()`. Așadar, definiția "originală" (din cadrul clasei `A`) nu poate fi accesată în interiorul clasei `B` decât dacă ne referim la ea explicit prin intermediul operatorului `::`.

Accesarea clasei de bază

În exemplul anterior am utilizat o funcție a clasei de bază. În locul utilizării denumirii clasei de bază poate fi folosită denumirea specială `parent` care este o referință la clasa de bază definită în cadrul construcției **`extends`**.

Folosirea denumirii speciale este utilă în cazul în care ierarhia de clase se modifică. În acest caz este suficientă o singură modificare în cadrul construcției **`extends`**, fără a mai fi necesare modificări în interiorul clasei derivate.

Așadar, definiția clasei `B` poate fi rescrisă astfel:

```
class B extends A{
  function exemplu(){
    echo "Funcția redefinită<BR>\n";
    parent::exemplu();
  }
}
```

Serializarea obiectelor

Prin serializare se înțelege crearea unui șir de octeți care conține reprezentarea internă (binară) a variabilei respective. Așadar, serializarea permite "salvarea" valorilor unei variabilei. Dacă este serializat un obiect sunt salvate doar proprietățile acestuia (variabilele membre) și numele clasei din care face parte, nu și metodele deoarece funcțiile nu reprezintă valori. Pentru a serializa un obiect este utilizată funcția `serialize()` care returnează șirul de octeți care conține reprezentarea binară.

Pentru a deserializa un obiect se folosește funcția pereche `unserialize()`. Pentru ca o astfel de operație să funcționeze corect este necesară definirea clasei din care face parte obiectul respectiv.

Funcția returnează valoarea variabilei serializate. În continuare vom prezenta un exemplu care ilustrează modul în care poate fi serializat și deserializat un obiect. Șirul de octeți obținut în urma serializării va fi scris într-un fișier și va fi citit din fișierul respectiv pentru efectuarea deserializării. De obicei serializarea și deserializarea sunt realizate în documente *PHP* diferite deoarece aceste operații nu au aproape nici o utilitate dacă sunt folosite în cadrul aceluiași document.

Documentul *PHP* în care se realizează serializarea trebuie să conțină o secvență asemănătoare cu următoarea:

```
class A{
  var $msg = "Hello World!";
  function scrie(){
    echo $this->msg;
  }
}
$a = new A;
$s = serialize($a);
//salvarea șirului într-un fișier
$fp = fopen("fisier", "w");
fputs($fp, $s);
fclose($fp);
```

Pentru deserializare al doilea document va conține următoarea secvență:

```
class A{
  var $msg = "Hello World!";
  function scrie(){
    echo $this->msg;
  }
}
//citirea șirului din fișier
$s = implode("", @file("fisier"));
$a = unserialize($s);
//după deserializare obiectul poate fi folosit
$a->show_one();
```

Referințe în PHP

Referințele pot fi utilizate pentru a accesa conținutul unei variabile folosind mai multe nume. Spre deosebire de limbajul *C*, în *PHP* referințele nu sunt pointeri, ci *alias*-uri într-o tabelă de simboluri.

În *PHP* denumirile variabilelor și conținutul acestora nu sunt unul și același lucru. Așadar este posibil ca același conținut să aibă denumiri diferite.

Utilizarea referințelor

Referințele *PHP* permit unor variabile cu denumiri diferite să corespundă unui același conținut. Cu alte cuvinte, instrucțiunea `$a = &$b` are ca efect faptul că `$a` și `$b` referă aceeași variabilă.

În această situație `$a` și `$b` au același statut. Nu se poate spune că `$a` referă `$b` sau invers.

O altă posibilitate de utilizare a referințelor este transmiterea prin referință a parametrilor unei funcții. Efectul unei astfel de transmisii este crearea unei variabile locale care referă spre același conținut ca variabila din contextul apelant.

Să considerăm următorul exemplu:

```
function inc(&$var){
  $var++;
}
$a=5;
inc($a);
```

Inițial valoarea variabilei `$a` este `5`. După apel variabila locală `$var` și variabila din contextul apelant `$a` indică spre același conținut. Valoarea păstrată în locația de memorie respectivă este incrementată (devine `6`) prin intermediul instrucțiunii `$var++;`. Datorită faptului că cele două variabile au același conținut, valoarea variabilei `$a` va fi `6` după executarea funcției.

Un parametru transmis prin referință poate fi:

- o variabilă;
- o instrucțiune **new**;
- o referință returnată de o funcție.

Dacă unei astfel de funcții i se transmite ca parametru un alt tip de expresie rezultatul este nedefinit. Așadar, pentru o funcție care are un parametru transmis prin referință nu se poate folosi o constantă în momentul apelului. De exemplu, pentru funcția `inc()` prezentată anterior nu este permis un apel de forma `inc(5)`.

Referințe globale

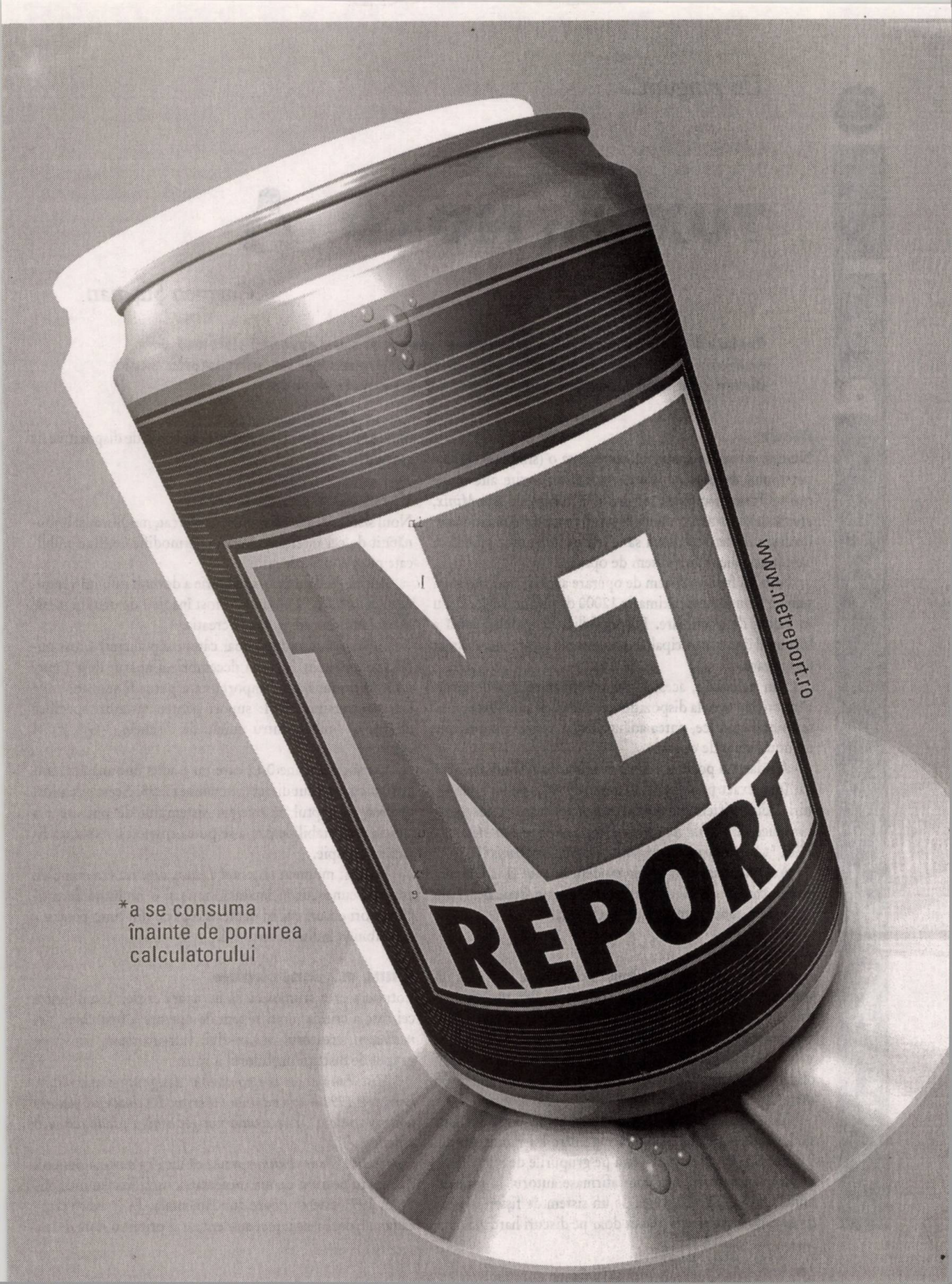
În momentul declarării unei variabile globale (printr-o instrucțiune de tipul **`global`** `$var`) se creează de fapt o referință spre o variabilă globală.

Cu alte cuvinte, această instrucțiune este echivalentă cu `$var = &$GLOBALS["var"];`.

Referința $this

În cadrul unei metode a unui obiect **`$this`** este întotdeauna o referință spre obiectul care utilizează funcția (obiectul curent).

Mihai Scorțaru este redactor-șef GInfo și poate fi contactat prin e-mail la adresa **skortzy@yahoo.com**. *Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa* **csoroiu@yahoo.com**.


NET
REPORT
www.netreport.ro
*a se consuma
înainte de pornirea
calculatorului

TUX din NORD

Carmen Şuşman

Probabil că mulţi dintre voi ştiu cum a apărut şi s-a dezvoltat sistemul de operare Linux. Pentru ceilalţi vom încerca să prezentăm pe scurt istoria unuia dintre cele mai cunoscute produse software apărute vreodată.

Minix

Nu este un secret faptul că *Linux* este o (simplă?) clonă a sistemului de operare *Unix*. Au existat multe alte clone mai mult sau mai puţin reuşite. Una dintre ele este *Minix*, creată de *Andrew S. Tanenbaum* în scopuri educaţionale. Profesorul olandez dorea să le arate studenţilor modul în care funcţionează un sistem de operare.

Minix a fost un sistem de operare al cărui cod sursă era public. Conţinea aproximativ 12000 de linii scrise în C sau în limbaj de asamblare. Disponibilitatea codului sursă a fost unul dintre principalele avantaje ale acestui nou sistem de operare.

Din nefericire, acest sistem de operare lăsa de dorit. Dar, oricine avea la dispoziţie cele 12000 de linii de cod putea modifica orice, putea adăuga orice; practic, putea crea un nou sistem de operare.

Aşadar, nu poate fi o surpriză faptul că, la un moment dat (mai exact pe data de 25 august 1991), pe un grup de ştiri dedicat sistemului de operare *Minix* a apărut un mesaj cu subiectul: "*What would you like to see most in Minix?*", adică "*Ce aţi dori cel mai mult să vedeţi în Minix?*".

Autorul mesajului era un student în anul II la Universitatea din *Helsinki*. Numele său, ***Linus Benedict Torvalds***, nu spunea nimic în acea perioadă.

Linux 0.01

În mesajul respectiv *Linus* spunea că doreşte să creeze un sistem de operare gratuit similar cu *Minix*, dar care nu conţine nici una dintre cele 12000 de linii deja celebre. El îi ruga pe toţi cei care doresc să îl ajute să îi spună ce le place şi ce nu le place la sistemul *Minix*. *Linus* spunea că acceptă orice propunere, dar nu promite că le va implementa pe toate.

Un grup de entuziaşti au agreat ideea şi *Linus* a început să dezvolte sistemul de operare pe care l-a numit *Linux*. Versiunea 0.01 a fost publicată pe grupurile de ştiri *Minix* în luna septembrie. Aşa cum afirmase autorul în mesajul iniţial, *Linux* 0.01 dispunea de un sistem de fişiere *multithreading*, dar putea fi folosit doar pe discuri hard *AT*, deoarece *Linus* avea la dispoziţie doar astfel de dispozitive de stocare.

Versiunile următoare

Noul sistem de operare a fost descărcat, modificat şi îmbunătăţit de cei interesaţi, versiunile modificate fiind publicate pe grupurile de ştiri.

Astfel, pe data de 5 octombrie a devenit publică versiunea *Linux* 0.02. "Lansarea" a fost însoţită de un nou mesaj al lui *Linus* în care îşi lăuda creaţia.

La scurt timp (după doar câteva săptămâni) a fost publicată versiunea 0.03. În decembrie a apărut deja *Linux* 0.10. O îmbunătăţire importantă a putut fi sesizată după lansarea versiunii 0.11: suport pentru tastaturi specifice diferitelor limbi, pentru unităţi de dischete, *VGA*, *EGA* etc.

După o versiune 0.12 care nu a adus îmbunătăţiri importante s-a trecut direct la versiunea 0.95. Acest salt a fost motivat de faptul că nucleul sistemului de operare era suficient de stabil pentru a se putea spune că versiunea finală se apropie.

În acest moment sistemul *Linux*, deşi nu era complet, era deja cunoscut în întreaga lume şi o mulţime de programatori entuziaşti îşi dedicau o parte din timp pentru a contribui la îmbunătăţirea lui.

Linus vs. Tanenbaum

Totul era prea frumos ca să nu apară critici. Unul dintre cei care a criticat noul sistem de operare a fost chiar *Tanenbaum*, creatorul *Minix*-ului. Într-un mesaj trimis pe grupul de discuţii profesorul a spus:

"*I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design* :-)".

Aşadar, *Tanenbaum* spunea că încă îşi susţine punctul de vedere potrivit căruia proiectarea unui nucleu monolitic în 1991 este o eroare fundamentală. El îl avertiza pe *Linus* că, dacă i-ar fi fost student, ar fi primit o notă destul

de mică pentru un astfel de proiect. Mai mult, în același mesaj *Tanenbaum* spunea că *Linux* este deja un sistem de operare depășit.

Susținut de întreaga comunitate nou-formată, *Linus* i-a dat un răspuns destul de îndrăzneț, devenit și el celebru:

"*Your job is being a professor and researcher: That's one hell of a good excuse for some of the brain-damages of Minix.*"

Tanenbaum era acuzat de faptul că își folosește poziția de profesor și cercetător ca o scuză pentru inconvenientele sistemului de operare *Minix*. Trebuie remarcat faptul că *Minix* nu avea un nucleu propriu-zis.

O evoluție surprinzătoare

În ciuda unor astfel de critici, comunitatea *Linux* a continuat să lucreze la produsul favorit. În scurt timp numărul programatorilor implicați în dezvoltarea sistemului de operare a ajuns mai mare decât o sută. Apoi, au devenit mii, apoi sute de mii...

A sosit timpul ca întreaga lume să se poată bucura de creația acestor entuziaști. *Linux* a devenit disponibil pentru oricine, codul sursă a fost distribuit gratuit și oricine îl putea copia, studia sau modifica.

Era inevitabil ca acest succes să atragă atenția comercianților. Aceștia au creat distribuții *Linux* care erau vândute chiar dacă sursele erau gratuite. Totuși, interesul acordat de aceștia a contribuit la impunerea sistemului de operare.

În 1991 *Linux* avea 10.000 de linii de cod și un singur utilizator. Un an mai târziu existau deja peste o mie de utilizatori, iar în 1993 numărul acestora a depășit 20000. În acest moment există zeci de milioane de utilizatori, iar numărul liniilor de cod este de ordinul milioanelor.

Linus Torvalds

Acum *Linux* este, cu siguranță, cel mai cunoscut sistem de operare distribuit gratuit. Unii susțin și că este cel mai bun. În orice caz, distribuțiile *Linux* sunt rivalii cei mai de temut pentru sistemele de operare *Microsoft*.

Pinguinul

La început *Linux* nu avea o siglă; acum îl are pe *Tux*, un pinguin simpatic al cărui nume provine de la numele autorului: *Torvalds' UniX*. Totuși, istoria acestui pinguin este aproape la fel de interesantă ca cea a sistemului de operare.

Într-o vacanță petrecută în sud *Linus* a întâlnit un pinguin pe care a dorit să-l mângâie. Creatura s-a speriat și l-a mușcat de mână.

După acest eveniment, potrivit declarațiilor sale, *Linus* s-a îmbolnăvit de o boală rară numită "*penguinitis*". Efectele acestei boli sunt destul de bizare: "*bolnavul se trezește noaptea și se gândește la pinguini, simțind chiar o dragoste paternă pentru aceștia*".

Așadar, din aceste motive sigla sistemului de operare *Linux* a devenit pinguinul. Motivul pentru care pinguinul este așezat este explicat tot de *Torvalds*: "*Ar trebui să vă imaginați un pinguin un pic supraponderal, stând jos și având un zâmbet de mulțumire după ce a mâncat câteva galoane de pește crud.*"

Carmen Șușman este redactor-șef adjunct al GInfo și poate fi contactată prin e-mail la adresa **schummy@email.ro**.

Distribuții Linux

Cele mai cunoscute distribuții:

- Redhat Linux (http://www.redhat.com)
- Mandrake Linux (http://www.linux-mandrake.com)
- SuSE Linux (http://www.suse.com)
- SlackWare Linux (http://www.slackware.com)
- Turbolinux (http://www.turbolinux.com)
- Debian/GNU Linux (http://www.debian.org)
- United Linux (http://www.calderasystems.com)
- Gentoo (http://www.gento.org)

Alte distribuții:

ASP Linux (http://www.asp-linux.com)
Beehive Linux (http://www.beehive.nu)
Conectiva Linux (http://www.conectiva.com)
e-smith (http://www.e-smith.com)
Kondara/MNU Linux (http://www.kondara.org)
Rock Linux (http://www.rocklinux.org)
SOT Linux (http://www.sot.com/en)
Ark Linux (http://www.arklinux.org)
DeLi Linux (http://delilinux.berlios.de)
Desktop/LX (http://www.lycorix.com)
ELX (Everyone's Linux) (http://www.elxlinux.com)
EvilEntity Linux (http://undeadlinux.com)
IcePack Linux (http://www.icepack-linux.com)
Independence (http://www.independence.seul.org)
LibraNet GNU/Linux (http://www.libranet.com)
MIZI Linux (http://www.mizi.com/en)
WinLinux (http://www.winlinux.net)
WinSlack (http://greboguru.org/winslack/index.html)
Xandros Linux (http://www.xandros.net)

COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui episod al serialului dedicat compresiilor de date vă vom prezenta metoda de compresie cu dicționare Lempel Ziv, cea mai rapidă metodă de compresie, și variante ale acesteia.

Metoda de compresie *LZ* (*Lempel-Ziv*) a fost implementată pentru prima dată de către ***Abraham Lempel*** și ***Jacob Ziv*** în anul 1977 și această implementare este denumită *LZ77*. Ulterior metoda a suferit unele modificări și au apărut variantele *LZ78*, *LZW* (*Lempel-Ziv-Welch* - 1984), *LZSS* (*Lempel-Ziv-Storer-Szymanski* - 1986).

Metoda LZ77

Metoda de compresie *LZ77* constă în păstrarea ultimelor n simboluri generate de o sursă de informație S și găsirea celei mai lungi secvențe de lungime maximă L_s ($L_s < n$) de simboluri generate de sursa S în cadrul secvenței de n simboluri stocate.

Algoritmul de compresie

Algoritmul de compresie pentru varianta *LZ77* este foarte simplu. La început avem un șir T care conține n simboluri. Fiecare dintre cele n simboluri se inițializează cu primul simbol al alfabetului unei surse de informație S. În continuare, se păstrează primele L_S simboluri generate de sursa de informație S pe ultimele L_S poziții din șirul T. Cele L_S simboluri se transmit la ieșirea E.

La pasul următor avem un șir w inițial de lungime 0. Atâta timp cât șirul w se regăsește în șirul T acesta este concatenat cu următorul simbol generat de sursa de informație S. Vom nota prin w_i șirul obținut din primele i simboluri ale șirului w. Fie i lungimea lui w în momentul în care acesta nu se mai regăsește în șirul T; aceasta înseamnă că șirul w_{i-1} este cel mai lung șir generat de sursa S care se află în șirul T începând de la poziția p. La ieșirea E se transmite poziția p, numărul (i - 1) și simbolul c de pe ultima poziție a șirului w, ceea ce se explică prin faptul că *următoarele i elemente ale șirului inițial se obțin din i - 1 simboluri consecutive din șirul T începând de la poziția p și simbolul c*. Șirul T se deplasează la stânga cu i poziții, iar ultimele i poziții vor fi umplute cu simbolurile șirului w. Acest pas se execută până când sursa de informație S nu mai generează simboluri.

Pentru a descrie în *pseudocod* algoritmii din cadrul acestui articol, vom nota cu '+' operația de concatenare dintre două șiruri de simboluri sau dintre un șir de simboluri și un caracter, vom considera că prima poziție a unui șir de simboluri este `0` și vom avea nevoie de următorii subalgoritmi:

- `CitesteSir(S, I)` - citește de la sursa de informație `S` un șir de maxim `I` simboluri pe care îl returnează ca rezultat;
- `CitesteSim(S)` - citește de la sursa de informație `S` un singur simbol pe care îl returnează ca rezultat;
- `CitesteNum(S)` - citește de la sursa de informație `S` un șir de simboluri pe care îl convertește la un număr care este returnat ca rezultat;
- `CitesteBit(S)` - citește de la sursa de informație `S` un șir de simboluri pe care îl convertește la o valoare logică (`0` sau `1`) care constituie rezultatul subalgoritmului;
- `Generează(S)` - returnează ca rezultat valoarea logică `adevărat` dacă sursa `S` mai generează simboluri și valoarea `fals` în caz contrar;
- `Transmite(E, I1, ..., In)` - transmite la ieșirea `E` conținutul variabilelor I_1, ..., I_n;
- `Lungime(w)` - returnează ca rezultat lungimea șirului de simboluri `w`;
- `Primele(s, i)` - returnează ca rezultat șirul de simboluri obținut din primele `i` simboluri ale șirului `s`;
- `Ultimele(s, i)` - returnează ca rezultat șirul de simboluri obținut din ultimele `i` simboluri ale șirului `s`;
- `Găsit(s, s1)` - returnează ca rezultat valoarea logică `adevărat` dacă șirul s_1 este subșir al șirului `s` și valoarea `fals` în caz contrar;
- `Poziție(s, s1)` - returnează ca rezultat poziția de început a primei apariții a șirului s_1 în șirul `s` dacă șirul s_1 este subșir al șirului `s` și valoarea `-1` în caz contrar.

În continuare vom prezenta algoritmul de compresie în *pseudocod*:

```
se inițializează n cu o constantă
se inițializează Ls cu o constantă
se inițializează T
w ← CitesteSir(S, Ls)
Transmite(E, w)
T ← Primele(T, n - Ls) + w
dacă nu Generează(S) atunci
   ieșire
sfârșit dacă
cât timp Generează(S) execută
   w ← ""
   cât timp Generează(S) și Găsit(T, w) execută
      c ← CitesteSim(S)
      w ← w + c
   sfârșit cât timp
   i ← Lungime(w)
   p ← Poziție(T, Primele(w, i - 1))
   Transmite(E, p, i - 1, c)
   T ← Ultimele(T, n - i) + w
sfârșit cât timp
```

În figura 1 este ilustrat modul în care se realizează compresia unui șir de simboluri folosind metoda *LZ77*. Se poate observa foarte ușor felul în care este actualizat șirul de simboluri T și cum variază șirul w la fiecare pas. Pasul 0 reprezintă pasul de dinaintea instrucțiunii repetitive **cât timp**.

Alfabetul A= {a, b, c, d}
S = "abbaabbaababbaacdaabaaba"
T = "aaaaaaaa"; n = 8; L_S = 4

Pas	Valoare T	Valoare w	Ieșire (p, i, c)
0	"aaaaabba"	"abba"	"abba"
1	"aaaaabba"	"abbaa"	(4, 4, "a")
2	"bbaabbaa"	"bab"	(1, 2, "b")
3	"abbaabab"	"baac"	(2, 3, "c")
4	"ababbaac"	"d"	(0, 0, "d")
5	"babbaacd"	"aab"	(4, 2, "b")
6	"baacdaab"	"aaba"	(5, 3, "a")

Figura 1: Exemplu de compresie cu *LZ77*

Algoritmul de decompresie

Algoritmul de decompresie pentru metoda *LZ77* constă în interpretarea șirului comprimat de simboluri generate de o sursă de informație S.

Inițial se citesc L_S simboluri care se transmit și sunt stocate și pe ultimele L_S poziții ale șirului T de lungime n.

În continuare, la fiecare pas se citește un triplet (p, i, c), se transmit i simboluri din T începând cu poziția p, se transmite simbolul c, T se deplasează la stânga cu i + 1 poziții și ultimele i + 1 poziții din T se completează cu șirul obținut din cele i + 1 simboluri transmise.

În pseudocod acest algoritm este:

```
se inițializează n cu o constantă
se inițializează Ls cu o constantă
se inițializează T
w ← CitesteSir(S, Ls)
Transmite(E, w)
T ← Primele(T, n - Ls) + w
dacă nu Generează(S) atunci
   ieșire
sfârșit dacă
cât timp Generează(S) execută
   p ← CitesteNum(S)
   i ← CitesteNum(S)
   c ← CitesteSim(S)
   w ← Primele(Ultimele(T, n - p), i) + c
   Transmite(E, w)
   T ← Ultimele(T, n - i - 1) + w
sfârșit cât timp
```

Observații

Complexitatea algoritmului de compresie este $O(n \cdot nr)$, unde nr este numărul de simboluri generate de sursa de informație S, deoarece funcția de căutare a unui subșir într-un șir are complexitatea $O(n)$ și modificarea șirului T se realizează în maxim $O(n)$ pași, în funcție de algoritmul utilizat (dacă se utilizează un șir circular, modificarea poate avea loc în $O(1)$ pași).

Complexitatea algoritmului de decompresie este $O(nr)$ în cazul în care modificarea șirului T se face în $O(1)$ pași și construirea lui w se face la fiecare pas folosind $O(i)$ instrucțiuni.

În general șirul T de simboluri poartă numele de ***fereastră mobilă***, acest algoritm fiind cel mai simplu algoritm de compresie din clasa *algoritmilor de compresie cu ferestre mobile*. *Dicționarul* în cazul algoritmului *LZ77* este constituit chiar de fereastra mobilă T.

Din cauză că șirul T este format numai din ultimele n simboluri generate de sursa de informație S algoritmul *LZ77* nu este optim, deoarece nu mai apar referințe la codificări anterioare deci, dacă mai apare un șir de simboluri care trebuie codificate identic cu un șir codificat anterior, se întâmplă destul de des să nu fie codificat în aceeași manieră (folosind un singur triplet (p, i, c)) fapt care duce la creșterea lungimii codificării întregului șir de simboluri generate de sursa S. Din acest motiv, cercetătorii ***Abraham Lempel*** și ***Jacob Ziv*** au renunțat la fereastra mobilă și în anul 1978 au realizat o nouă implementare în care *dicționarul* de compresie era constituit de șirurile de simboluri codificate anterior.

LZ77 este un algoritm fundamental de compresie a datelor și stă la baza clasei de algoritmi de compresie cu fereastră mobilă, în timp ce *LZ78* este baza unei noi clase de algoritmi de compresie.

Metoda LZSS

În cazul metodei *LZ77*, se observă foarte ușor că dacă ultimul șir de simboluri w_{i-1} care se regăsește în șirul T este

chiar șirul vid atunci, pentru a reprezenta șirul w care se codifică se transmit numerele p și $(i - 1)$ care au valoarea 0 și simbolul din șirul w. Aceasta înseamnă că pentru a codifica un singur simbol trebuie două numere care sunt inutile.

Metoda *LZSS* constituie o îmbunătățire asupra metodei *LZ77*. Îmbunătățirea este aceea că în cazul enunțat anterior, dacă avem de codificat un singur simbol, se transmite bitul 0 urmat de simbol, iar dacă avem de codificat un șir mai lung de simboluri, se transmite bitul 1 urmat de perechea (p, i). În acest ultim caz, la iterația următoare șirul w va fi inițializat cu șirul format din ultimul simbol generat de sursa S.

În concluzie, dacă lungimea codificării unui șir este mai mare decât șirul necodificat, atunci se va transmite la ieșire necodificat, precedat de un bit setat pe 0. Se observă că în cazul acestui algoritm, dacă este nevoie să se codifice un șir, în fereastra mobilă se va introduce doar cel mai lung subșir care se regăsește în aceasta și nu primul subșir care nu se regăsește (cum se întâmplă în cazul algoritmului de compresie *LZ77*).

Algoritmul de compresie *LZSS* este:

```
se inițializează n cu o constantă
se inițializează Ls cu o constantă
se inițializează T
w ← CitesteSir(S, Ls)
Transmite(E, w)
T ← Primele(T, n - Ls) + w
dacă nu Generează(S) atunci
  ieșire
sfârșit dacă
  w ← ""
cât timp Generează(S) execută
   cât timp Generează(S) și Găsit(T, w) execută
     c ← CitesteSim(S)
     w ← w + c
   sfârșit cât timp
   i ← Lungime(w)
   dacă i = 1 atunci
      Transmite(E, 0, c)
      w' ← ""
   altfel
      p ← Poziție(T, Primele(w, i))
      dacă p = -1 atunci
         i ← i - 1
         p ← Poziție(T, Primele(w, i))
         w' ← c
         dacă i = 1 atunci
            Transmite(E, 0, Primele(w, 1))
         altfel
            Transmite(E, 1, p, i )
      sfârșit dacă
      altfel
         w' ← ""
         Transmite(E, 1, p, i )
      sfârșit dacă
      sfârșit dacă
      T ← Ultimele(T, n - i) + Primele(w, i)
      w ← w'
sfârșit cât timp
```

În figura următoare se poate observa modul de compresie a unui șir de simboluri generate de o sursă S, folosind algoritmul *LZSS*:

Alfabetul A= {a, b, c, d}
S = "abbaabbaabababbaacdaabaaba"
T = "aaaaaaaa"; $n = 8$; $L_S = 4$

Pas	Valoare T	Valoare w	Ieșire (1, p, i) sau (0, c)
0	"aaaaabba"	"abba"	"abba"
1	"aaaaabba"	"abbaa"	(1, 4, 4)
2	"abbaabba"	"aba"	(1, 0, 2)
3	"baabbaab"	"abbaac"	(1, 2, 5)
4	"aababbaa"	"c"	(0, "c")
5	"ababbaac"	"d"	(0, "d")
6	"babbaacd"	"aab"	(1, 4, 2)
7	"bbaacdaa"	"baab"	(1, 2, 3)
8	"acdaabaa"	"ba"	(1, 5, 2)

Figura 2: Exemplu de compresie cu *LZSS*

Decompresia unui șir de simboluri comprimat cu metoda *LZSS* se realizează la fel ca în cazul algoritmului *LZ77* cu precizarea că în momentul decodificării unei perechi se verifică bitul care o precede, și anume, dacă bitul este 0, atunci, la ieșire se va transmite un simbol c, iar dacă este 1, atunci, la ieșire se va transmite subșirul din fereastra mobilă care începe de la o poziție p și are lungimea i. Acțiunea de decodificare va fi urmată de actualizarea ferestrei mobile.

În continuare prezentăm varianta pseudocod a algoritmului de decompresie *LZSS*:

```
se inițializează n cu o constantă
se inițializează Ls cu o constantă
se inițializează T
w ← CitesteSir(S, Ls)
Transmite(E, w)
T ← Primele(T, n - Ls) + w
dacă nu Generează(S) atunci
  ieșire
sfârșit dacă
cât timp Generează(S) execută
  b ← CitesteBit(S)
dacă b = 0 atunci
    c ← CitesteSim(S)
    w ← c
    i ← 1
altfel
```

```
    p ← CitesteNum(S)
    i ← CitesteNum(S)
    w ← Primele(Ultimele(T, n - p), i)
  sfârșit dacă
  Transmite(E, w)
  T ← Ultimele(T, n - i) + w
sfârșit cât timp
```

Ordinele de complexitate ale algoritmilor de compresie și decompresie pentru metoda *LZSS* sunt aceleași cu cele ale algoritmilor de compresie, respectiv decompresie, pentru algoritmul *LZ77*, în schimb, un șir de simboluri generat de o sursă de informație *S* este comprimat mai bine dacă se folosește algoritmul *LZSS*.

Metoda LZ78

Spre deosebire de cele două metode de compresie a datelor prezentate anterior, metoda *LZ78* nu folosește o fereastră mobilă, ci reține toate șirurile de simboluri codificate anterior într-o listă.

La începutul compresiei, mărimea listei de elemente `T` codificate anterior este 1 și conține pe poziția 0 șirul vid. La un pas, se consideră lista elementelor obținută la pasul anterior și se citește de la o sursă de informație *S* cel mai scurt șir de simboluri `w` care nu se află în lista elementelor `T`. În această listă se va adăuga șirul `w`, iar la ieșirea `E` se va transmite o pereche formată din numărul de ordine al primei apariții a șirului obținut din șirul `w` prin eliminarea ultimului simbol din acesta.

Pentru a putea prezenta algoritmul de compresie *LZ78* avem nevoie de subalgoritmul `Caută(s, w)`, care primește ca parametri o listă de șiruri de simboluri `s` și un șir de simboluri `w`, returnând ca rezultat numărul de ordine al primei apariții a șirului `w` în lista `s`. Dacă `w` nu apare printre elementele listei `s`, atunci rezultatul subalgoritmului va fi -1. Vom considera că numărul de ordine al primului șir care apare în lista `s` este 0.

```
T[0] ← w              //inițial, lista T conține doar șirul vid
k ← 1                 //k reprezintă mărimea listei T
cât timp Generează(S) execută
  w ← ""
  cât timp Generează(S) și Caută(T, w) ≥ 0 execută
    c ← CitesteSim(S)
    w ← w + c
  sfârșit cât timp
  T[k] ← w
  k ← k + 1
  i ← Lungime(w)
  w ← Primele(w, i - 1)
  i ← Caută(T, w)
  Transmite(E, i, c)
sfârșit cât timp
```

În practică, atunci când dimensiunea listei `T` ajunge la o valoare prestabilită (datorită insuficienței resurselor de memorie) nu se mai adaugă elemente în aceasta, acest fapt ducând la ineficiența algorimtului pentru date foarte multe și pentru multe apariții ale unor șiruri de simboluri care nu se află printre elementele listei.

În figura 3 se poate observa modul în care se realizează compresia datelor folosind algorimtul prezentat anterior.

Alfabetul A= {a, b, c, d}
S= "abbaabbaababbaacdaabaaba"
T= [""]

Pas	Valoare w	Indice element adăugat în T	Ieșire (i, c)
1	"a"	1	(0, "a")
2	"b"	2	(0, "b")
3	"ba"	3	(2, "a")
4	"ab"	4	(1, "b")
5	"baa"	5	(3, "a")
6	"bab"	6	(3, "b")
7	"baac"	7	(5, "c")
8	"d"	8	(0, "d")
9	"aa"	9	(1, "a")
10	"baab"	10	(5, "b")
11	"a"	11	(0, "a")

Figura 3: Exemplu de compresie cu *LZ78*

Algoritmul de decompresie devine acum foarte simplu deoarece, la fiecare pas, se interpretează o pereche (`i`, `T`) care se citește de la sursa de informații și se transmite șirul care se află pe poziția `i` în lista `T` urmat de caracterul din pereche.

Inițial, ca și în cazul compresiei, lista `T` conține doar șirul vid. La fiecare pas se adaugă în lista `T` șirul format din elementul de pe poziția `i` din `T` și din caracterul care a fost citit.

În pseudocod, algoritmul de decompresie *LZ78* este:

```
T[0] ← w              //inițial, lista T conține doar șirul vid
k ← 1                 //k reprezintă mărimea listei T
cât timp Generează(S) execută
  i ← CitesteNum(S)
  c ← CitesteSim(S)
  Transmite(E, T[i], c)
  T[k] ← T[i] + c
  k ← k + 1
sfârșit cât timp
```

Complexitatea algoritmului de compresie are ordinul $O(m \cdot n)$, unde m reprezintă media lungimii șirurilor de simboluri care se află în lista `T`, în cazul în care se folosește un algoritm banal de căutare, iar n reprezintă numărul total de simboluri generate de o sursă de informație *S*. În cazul în care se folosește un algoritm performant de căutare și lista `T` este reprezentată folosind *arbori de sufixe*, atunci această complexitate scade foarte mult.

Algoritmul de decompresie are ordinul de complexitate $O(n)$, unde n reprezintă numărul de simboluri care trebuie decodificate.

Metoda LZW

Welch a observat o deficiență a algoritmului *LZ78*, ajungând la concluzia că pentru a decodifica un șir de simboluri nu este nevoie de o pereche formată dintr-un indice și un simbol.

Prin urmare, acesta a modificat algoritmii de compresie și decompresie astfel:

- lista T era inițializată cu șiruri de simboluri, fiecare șir de simboluri fiind format din câte un simbol al alfabetului sursei de informație S. În lista T sunt atâtea șiruri câte simboluri are alfabetul;
- la fiecare pas din iterație:
  - șirul w este inițializat cu ultimul simbol al șirul w obținut la pasul anterior;
  - în lista T se adaugă cel mai scurt șir w, obținut prin concatenări succesive cu simboluri generate de sursa S, care nu se alfă în T;
  - la ieșire se transmite un singur cod care reprezintă numărul de ordine al apariției lui w din care se extrage ultimul simbol.

Acest algoritm este cunoscut sub denumirea *LZW*.

În continuare prezentăm în *pseudocod* algoritmul care realizează compresia unui șir de simboluri generate de o sursă de informație S pentru metoda *LZW*. În figura 4 se poate observa modul în care se realizează compresia unui șir de simboluri.

```
se inițializează T cu cele m simboluri ale alfabetului sursei S
k ← m                          //k reprezintă mărimea listei, T
w ← ""
cât timp Generează(S) execută
  cât timp Generează(S) și Caută(T, w) ≥ 0 execută
    c ← CitețteSim(S)
    w ← w + c
  sfârșit cât timp
  i ← Caută(T, w)
  dacă i ≥ 0 atunci   //sursa S nu mai generează simboluri, sfârșitul compresiei
    Transmite(E, i)
  altfel
    T[k] ← w
    k ← k + 1
    i ← Lungime(w)
    w ← Primele(w, i - 1)
    i ← Caută(T, w)
    Transmite(E, i)
    w ← c
  sfârșit dacă
sfârșit cât timp
```

Pe exemplul din figura 4 se poate observa că pentru a comprima șirul se transmit 14 coduri, spre deosebire de algoritmul *LZ78* care, pentru a comprima același șir de simboluri, transmite 11 coduri și 11 simboluri.

Alfabetul A= {a, b, c, d}
S = "abbaabbaabababbaacdaabaaba"
T = ["a", "b", "c", "d"]
k = 4

Pas	Valoare w	Indice element adăugat în T	Ieșire (i)
1	"ab"	4	(0)
2	"bb"	5	(1)
3	"ba"	6	(1)
4	"abb"	7	(4)
5	"baa"	8	(6)
6	"aba"	9	(4)
7	"abba"	10	(7)
8	"aa"	11	(0)
9	"ac"	12	(0)
10	"cd"	13	(2)
11	"da"	14	(3)
12	"aab"	15	(11)
13	"baab"	16	(8)
14	"ba"		(6)

Figura 4: Exemplu de compresie cu *LZW*

Algoritmul de decompresie este asemănător cu cel de la metoda *LZ78* cu deosebirea că în lista T se adaugă șirul format din șirul decodificat la pasul anterior și primul simbol al șirului decodificat la pasul curent.

În *pseudocod*, acest algoritm este:

```
se inițializează T cu cele m simboluri ale alfabetului sursei S
k ← m                          //k reprezintă mărimea listei, T
i ← CitețteNum(S)
w ← T[i]
Transmite(E, w)
cât timp Generează(S) execută
  i ← CitețteNum(S)
  T[k] ← w + Primele(T[i], 1)
  w = T[i]
  Transmite(E, T[i])
  k ← k + 1
sfârșit cât timp
```

Ordinele de complexitate ale algoritmilor de compresie și decompresie ale metodei *LZW* sunt aceleași ca și în cazul metodei *LZ78*.

Algoritmii *LZ78* și *LZW*, datorită rapidității, sunt utilizați des în programele comerciale de compresie singurul inconvenient fiind acela că mărimea listei T trebuie limitată datorită cantității mici de resurse disponibile pe sistemele de calcul.

Claudiu Soroiu este redactor al GInfo și poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

TEOREME de programare

Clara Ionescu

Continuăm prezentarea unor algoritmi simpli care vor conferi programelor claritate, iar programatorilor siguranța unei rezolvări corecte.

Partiționarea unui șir în două subșiruri

În numărul precedent am rezolvat câteva probleme în care se dădea un șir și se cerea selecționarea mai multor elemente, având o aceeași proprietate dată. Se pune întrebarea: ce se întâmplă cu elementele neselecționate? În acest episod al serialului nostru vom da răspuns la această întrebare.

P21. Se consideră *n* numere naturale. Să partiționăm șirul în două subșiruri astfel încât unul să conțină numerele pare, iar celălalt numerele impare!

P22. Să se creeze subșirul fetelor, respectiv al băieților pe baza șirului numelor elevilor unei clase!

P23. Cunoscând mediile anuale dintr-un catalog, să se creeze subșirul elevilor care au media mai mare decât 9, din cei rămași subșirul celor cu media mai mare decât 8 și așa mai departe!

Prima particularitate pe care o observăm este că, de data aceasta, ***se dă un șir*** și ***se cer mai multe***. Practic, va trebui să efectuăm una după alta mai multe *selecționări*. De exemplu, în cazul problemei **P23**, mai întâi vom selecționa acele elemente care au prima proprietate, apoi din șirul elementelor puse deoparte (rămase după selecționare) selecționăm elementele având cea de a doua proprietate și așa mai departe. Rezultă că algoritmul constă din aplicarea repetată a *partiționării în două șiruri* a șirului dat. Dacă partiționarea se face în două subșiruri, este clar că elementele neselecționate pentru primul șir îl vor forma pe cel de al doilea.

Vom prezenta mai multe modalități de a realiza cerința acestui tip de problemă. În prima variantă elementele având proprietatea cerută și cele care nu au această proprietate le vom așeza în două șiruri noi. Aceste șiruri le vom declara având dimensiunea șirului dat, deoarece nu se poate anticipa numărul exact de elemente. Este posibil ca toate elementele să fie selecționate în primul șir sau nici unul. În *pseudocod*, lucrăm cu următoarele entități:

n: întreg {*numărul elementelor șirului de prelucrat*}
X: tablou(1..n): tip element {*elementele șirului dat*}
bucY, bucZ: întregi {*numărul elementelor subșirurilor*}
Y, Z: tablou(1..n): întreg { *elementele șirurilor noi*}

```
subalgoritm Partiționare1(n,X,bucY,Y,bucZ,Z):
  bucY ← 0
  bucZ ← 0
  pentru i ← 1, n execută
    dacă P(X[i]) atunci
      bucY ← bucY +1
      Y[bucY] ← X[i]
    altfel
      bucZ ← bucZ +1
      Z[bucZ] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Dacă am dori să economisim spațiul ripisit cu o astfel de soluție putem utiliza un singur șir. Elementele selecționate le reținem în acesta, așezându-le de la început spre sfârșit, iar pe cele rămase le așezăm de la ultimul element spre primul. Evident, nu există pericolul să "ne ciocnim", deoarece avem exact *n* elemente care trebuie așezate pe *n* locuri.

```
subalgoritm Partiționare2(n,X,bucY,Y):
  bucY ← 0
  pentru i ← 1, n execută
    dacă P(X[i]) atunci
      bucY ← bucY +1
      Y[bucY] ← X[i]
    altfel
      Z[n + 1 - i + bucZ] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Să observăm că această rezolvare ne furnizează șirul elementelor neselecționate în ordine inversă față de pozițiile elementelor sale în șirul dat. În cazul în care după partiționare nu avem nevoie de șirul dat în forma sa originală, partiționarea, exact ca selecționarea, poate fi realizată folosind spațiul alocat șirului dat, adică *pe loc*.

Ne putem imagina o soluție care ar folosi un algoritm conform căruia am avansa în șir până la descoperirea primului element care nu are proprietatea cerută și l-am interschimbat cu primul găsit care, în schimb, are proprietatea cerută. Această soluție nefiind suficient de eficientă, propunem următorul algoritm. Punem primul element deoparte păstrându-i valoarea într-o variabilă auxiliară și căutăm un element, având proprietatea cerută, pornind de la capătul șirului și îl punem pe locul eliberat la începutul șirului. Acum vom căuta, pornind de la această poziție, un element care nu are proprietatea cerută și îl așezăm pe locul eliberat la sfârșitul șirului. Continuăm acest procedeu până când, în urma avansării în cele două direcții, ne vom întâlni.

```
subalgoritm Partiționare3(n,X,buc):
  înc ← 1
  sf ← n
  aux ← X[înc]
  cât timp înc < sf
    cât timp (înc < sf) and not (P(X[sf]))
      sf ← sf - 1
    sfârșit cât timp
    dacă înc < sf atunci
      X[înc] ← X[sf]
      înc ← înc + 1
      cât timp (înc < sf) and (P(X[înc]))
        înc ← înc + 1
      sfârșit cât timp
      dacă înc < sf atunci
        X[sf] ← X[înc]
        sf ← sf - 1
      sfârșit dacă
    sfârșit dacă
  sfârșit cât timp
  X[înc] ← aux
  dacă P(X[înc]) atunci
    buc ← înc
  altfel
  buc ← înc - 1
  sfârșit dacă
sfârșit subalgoritm
```

Vom rezolva problemele enunțate la începutul articolului. În P21 se cere șirul numerelor pare și cel al numerelor impare luate dintr-un șir de numere naturale. Vom rezolva problema cu algoritmul Partiționare2 care reține cele două subșiruri în locul rezervat șirului dat.

Presupunem că în programul apelant am declarat tipul:

```
type tablou=array[1..50] of Word;
procedure P21(n:Byte; X:tablou; var buc:Byte;
                                  var Y:tablou);
begin
  buc:=0;
  for i:= 1 to n do
    if Odd(X[i]) then begin
      Inc(buc);
      Y[buc]:=X[i]
    end else
      Y[n+1-i+buc]:=X[i]
end;
```

Problema P22 este foarte asemănătoare cu P21, deci rezolvarea ei o lăsăm pe seama cititorului.

Să vedem cum se rezolvă problema P23. În cazul acestei probleme vom aplica algoritmul de partiționare de mai multe ori, șirul de partiționat la pasul doi fiind cel pus deoparte după prima partiționare. Pentru a economisi spațiu, nu vom declara mai multe șiruri, ci vom realiza partiționarea și de data aceasta pe loc, folosind algoritmul Partiționare3 cu parametri care se vor referi la șirul curent care trebuie partiționat.

Presupunem că avem declarate tipurile și variabilele:

```
type unelev=record
                nume:string[30];
                media:Real
              end;
     catalog=array[1..15] of unelev;
var n,i,primul,buc,limita:Byte;
    X:catalog;
procedure part(n,primul,limita:Byte; var
                          X:catalog; var buc:Byte);
var aux:unelev;
begin
  aux:=X[primul];
  while primul < n do begin
  while (primul < n) and
    not (X[n].media > limita) do Dec(n);
    if primul < n then begin
      X[primul]:=X[n];
      Inc(primul);
      while (primul < n) and
     (X[primul].media > limita) do Inc(primul);
      if primul < n then begin
        X[n]:=X[primul];
        Dec(n)
      end
    end
  end;
  X[primul]:=aux;
  if X[primul].media > limita then buc:=primul
  else buc:=primul - 1
end;
```

În programul apelant vom avea următoarea secvență în care apelăm repetat subprogramul part:

```
primul:=1       {primul indice al șirului curent}
limita:=9;{se caută medii mai mari decât limita}
while limita >= 5 do begin     { medii posibile }
  part(n,primul,limita,x,buc);
  primul:=buc+1;          {indicele de început al}
          {subșirului care trebuie partiționat}
  Dec(limita)          {următorul grup de medii}
end;
```

15 feb.-30 mai 2003

CUPA SIVECO

concurs național
de software educațional

Sub patronajul Ministerului Educației și Cercetării
Partener media PC Magazine România

SIVECO ROMANIA SA

Ministerul Educației și Cercetării

PC MAGAZINE ROMÂNIA

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

SPONSOR DE AUR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

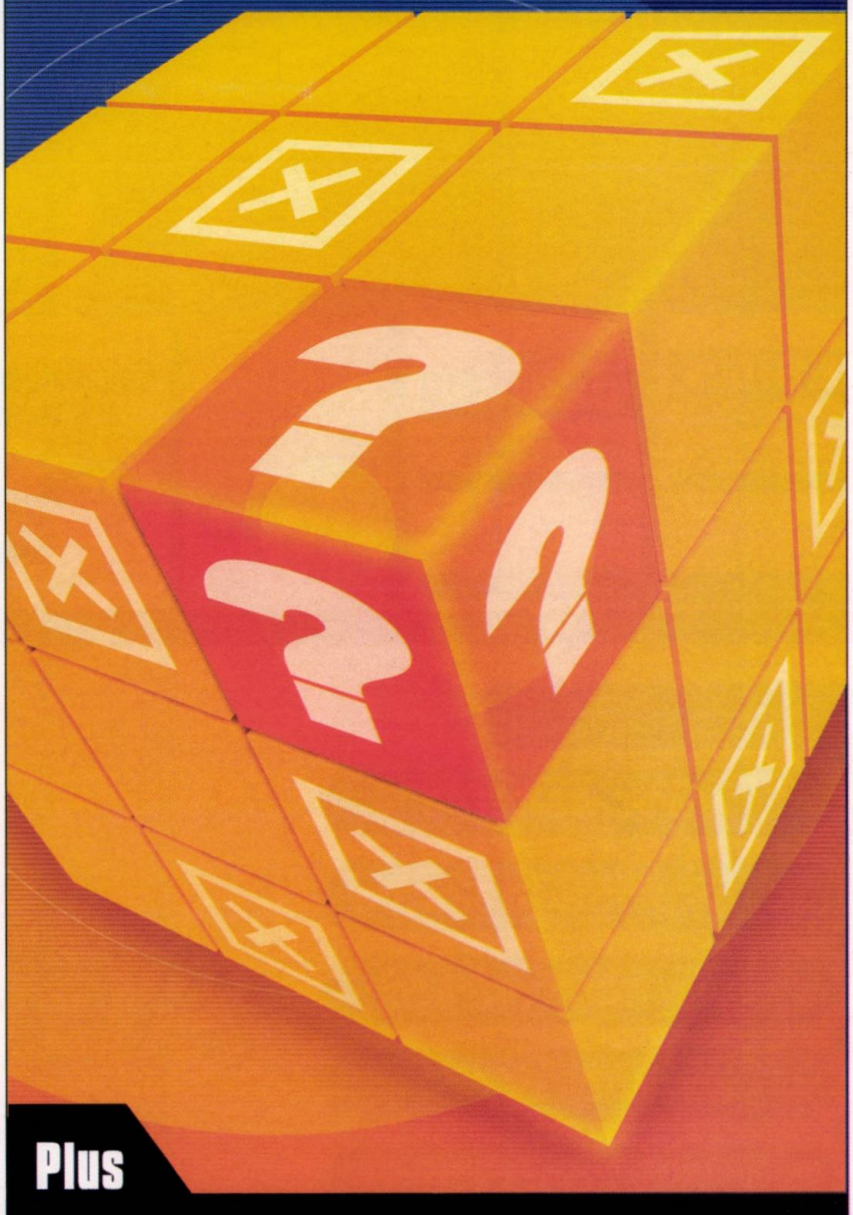
Aprilie 2003
Vol. 13/4

Lei 30.000

din sumarul numărului:

bac info 2003

Test grilă
neoficial



Plus

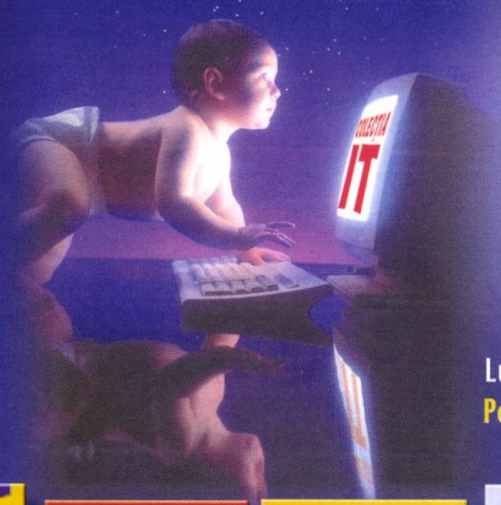
serial
compresia datelor

internet
pagini web cu php4

babel
despre concursuri

Editura CORINT

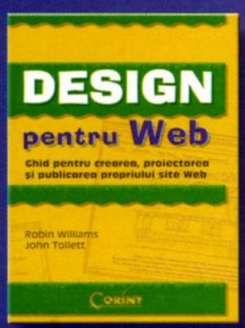
El a aflat...
Voi nu vreţi?

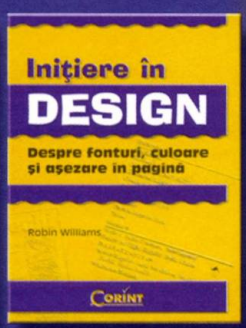


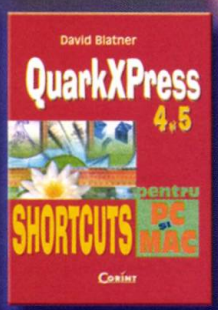
COLECŢIA
IT

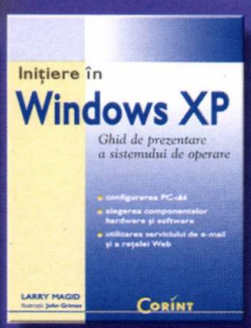
CORINT Difuzare: Str. Teodosie Rudeanu nr. 21;
Tel: 222 19 49; 223 19 28; Fax: 222 40 34;
E-mail: corint@dnt.ro

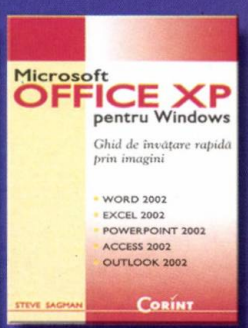
Lucrări sub licenţa prestigioaselor edituri:
Peachpit Press – Pearson Technology Group,
Random House Webster's











www.agora.ro
www.pcmagazine.ro
www.netreport.ro
www.eweek.ro
www.pcconcrete.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Vol. 13, Nr. 4 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Redactor-șef:
Mihai Scorțaru
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Claudiu Soroiu
A mai colaborat:
Mihai Stroe (București)
Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/192.422, 0265/166.516
Fax: 0265/166.290
http://www.ginfo.ro
ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 0265/162.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*", "*NetReport*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
Florin Oprescu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

probleme.. PAG. 5

Faza JUDEȚEANĂ a olimpiadei de INFORMATICĂ pentru GIMNAZIU

Faza JUDEȚEANĂ a olimpiadei de INFORMATICĂ pentru LICEE

~ subiecte unice în ~
~ întreaga țară ~

concurs.. PAG. 13

Despre desfășurarea rundelor #13-#16

Clasamentul general după 16 etape

Enunțurile problemelor de la rundele #17-#20

babel.. PAG. 21

Despre CONCURSURI

Mihai STROE

~ cele mai importante ~
~ concursuri de ~
~ programare ~

~ pregătirea în vederea ~
~ participării la concursuri ~

bac.. PAG. 28

BAC INFO' 2003

~ un test grilă neoficial ~
~ pentru examenul de ~
~ Bacalaureat ~

internet.. PAG. 36

Pagini WEB cu PHP 4

Mihai SCORȚARU
Claudiu SOROIU

serial.. PAG. 42

COMPRESIA datelor

Claudiu SOROIU

~ transformarea datelor ~
~ pentru a permite o ~
~ rată de compresie ~
~ mai mare fără ~
~ pierderi de ~
~ informație ~

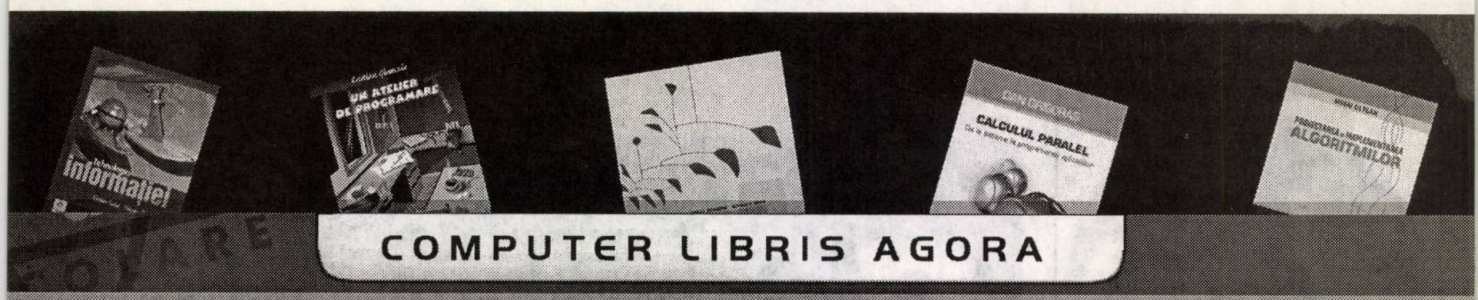
begin.. PAG. 47

TEOREME de programare

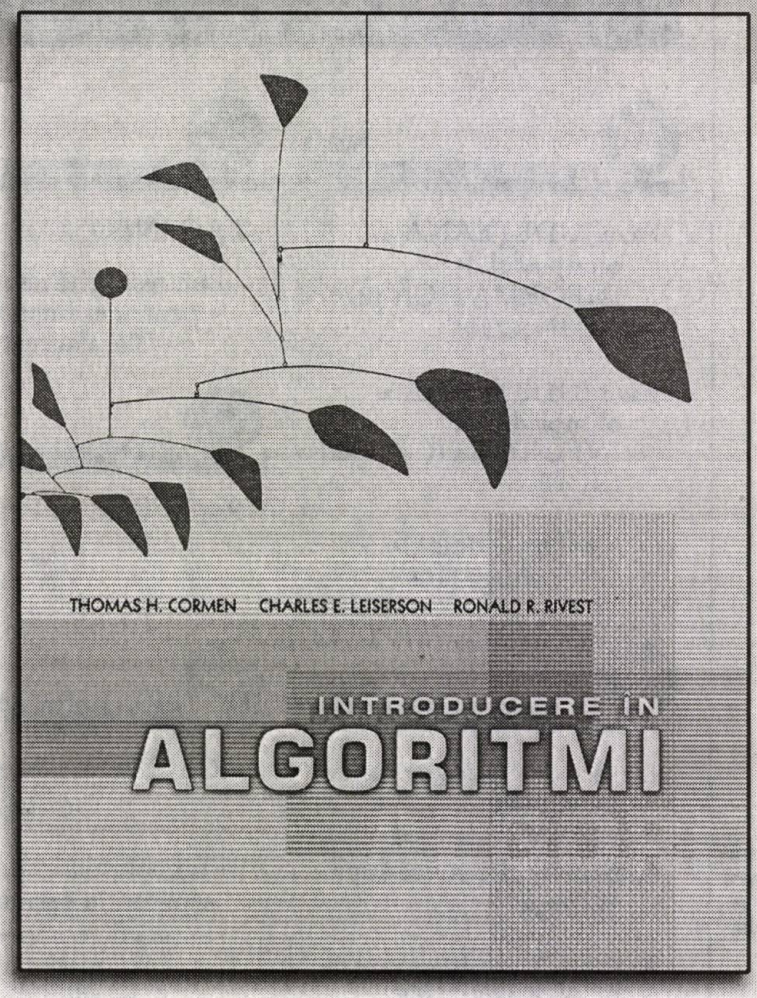
Clara IONESCU

~ simularea operațiilor cu ~
~ mulțimi folosind tablouri ~
~ unidimensionale ~

~ intersecție ~
~ reuniune ~



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

OLIMPIADA și contestațiile

În această perioadă au loc majoritatea olimpiadelor pentru elevii de liceu și gimnaziu. Începând cu luna ianuarie, elevii mai mult sau mai puțin talentați își încearcă norocul la astfel de concursuri.

Unii vor să demonstreze că sunt cei mai buni, altora le place să participe, alții sunt doar curioși și o mare parte profită de ocazie pentru a lega prietenii.

În urma concursurilor, unii se califică, alții rămân cu experiența acumulată. Unii sunt fericiți știind că au reușit să-și arate calitățile, alții sunt triști deoarece consideră că rezultatele nu reflectă realitatea.

În ultima perioadă am observat că mulți elevi depun contestații "din principiu". Motivele sunt cel puțin bizare, dar ideea este: "poate îmi mai dau câteva puncte".

Am observat că în cazul disciplinelor reale numărul contestațiilor este mult mai mare decât în cazul celor umane deși, datorită subiectivismului, s-ar părea că ar trebui să fie invers.

Nu știu dacă informatica reprezintă un caz special, dar aici se depun, cu siguranță, cele mai multe contestații. La mai multe concursuri la care am participat, peste jumătate dintre participanți au fost nemulțumiți.

Principalul motiv este, în opinia mea, evaluarea automată a problemelor; de fapt, nu metoda în sine, ci modul în care este ea pusă în practică la noi... Dacă elevul nu este de față, în cel mai fericit caz, el primește un borderou cu rezultatele și promisiunea că își va putea verifica ulterior programul. Din nefericire, de obicei, promisiunea rămâne doar promisiune...

Dacă elevul este lângă profesorul care evaluează, atunci vede pe ecran o mulțime de texte și, eventual, niște fișiere care nu îi spun mare lucru.

Un alt motiv al contestațiilor este strictețea evaluării. De exemplu, dacă la matematică scrii jumătate din rezolvarea unei probleme, atunci primești aproximativ jumătate din punctaj; la informatică o jumătate de program duce la o "banală" *eroare de compilare: 0 puncte*. Tot aici intră problemele legate de unit-ul *crt*, așteptarea reacțiilor de la tastatură, nerespectarea denumirilor sau a locațiilor fișierelor etc.

Nu se poate modifica nimic referitor la acest aspect al evaluării (dacă am puncta sursele ne-am întoarce cu zece ani în urmă); totuși, problema nu este cauzată de modul în care se desfășoară evaluarea, ci de faptul că elevii nu au la dispoziție suficiente informații referitoare la modul în care funcționează un astfel de program de evaluare. Nici unii profesori nu știu exact cum se evaluează un program, așa că nu mă mai mir atunci când elevul mă întreabă de ce nu apăs *Enter*.

La faza locală a olimpiadei de informatică, în județul *Cluj*, a fost folosit sistemul de evaluare creat de redacția revistei noastre. Nu prea aveam cum să apăs *Enter*, deoarece fereastra în care rula programul elevului era ascunsă. Încercând să îi explic că datorită faptului că programul său așteaptă apăsarea unei taste, limita de timp e depășită, mi s-a explicat (cu lux de amănunte) că nu am dreptate; după părerea elevului, timpul trebuia măsurat până înainte ca programul să execute instrucțiunea prin care se așteaptă apăsarea tastei deoarece, de fapt, programul nu mai face nimic, deci nu se mai execută etc.

Este destul de clar că elevul nu știa nici ce înseamnă timpul de execuție al unui program. Nici nu prea avea cum să știe, având în vedere că nu i-a spus nimeni. În acest caz, elevul a lipsit de la întâlnirea care s-a organizat în liceul respectiv pentru ca elevii să fie familiarizați cu aceste evaluatoare. Așadar, un punct în plus pentru liceu (*Liceul Emil Racoviță* din *Cluj*) și 0 puncte pentru elev.

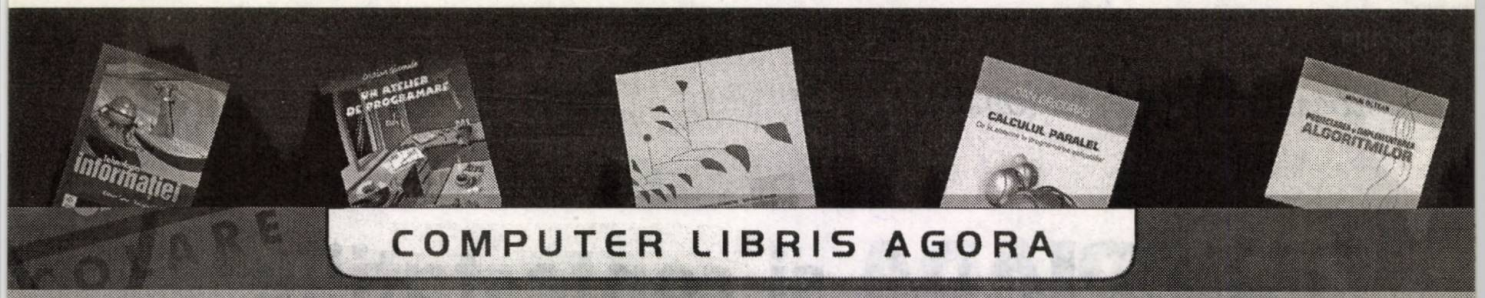
Din nefericire, în majoritatea celorlalte licee astfel de întâlniri nu au avut loc. Cei care au avut noroc nu au pierdut puncte din greșeli stupide, dar au fost mulți (prea mulți) cei care au pierdut punctele pe care le-ar fi meritat datorită faptului că nu le-a explicat nimeni cum trebuie scris un program la olimpiadă.

Cel mai trist motiv al contestațiilor îl reprezintă greșelile comisiei. Din nefericire, la informatică acestea sunt mult prea dese. La olimpiada județeană din acest an, au fost comise unele erori care au stârnit zâmbete. Am vrut să scriu un articol cu titlul *După OJI 2003* în care să prezint derularea concursului, dar am renunțat deoarece am observat că articolul *După ONI 2002* pe care l-am publicat în numărul 12/8 (decembrie 2002) al revistei nu a avut nici un efect. Au fost comise aceleași tipuri de greșeli, deci cine ar fi trebuit să audă nu a auzit.

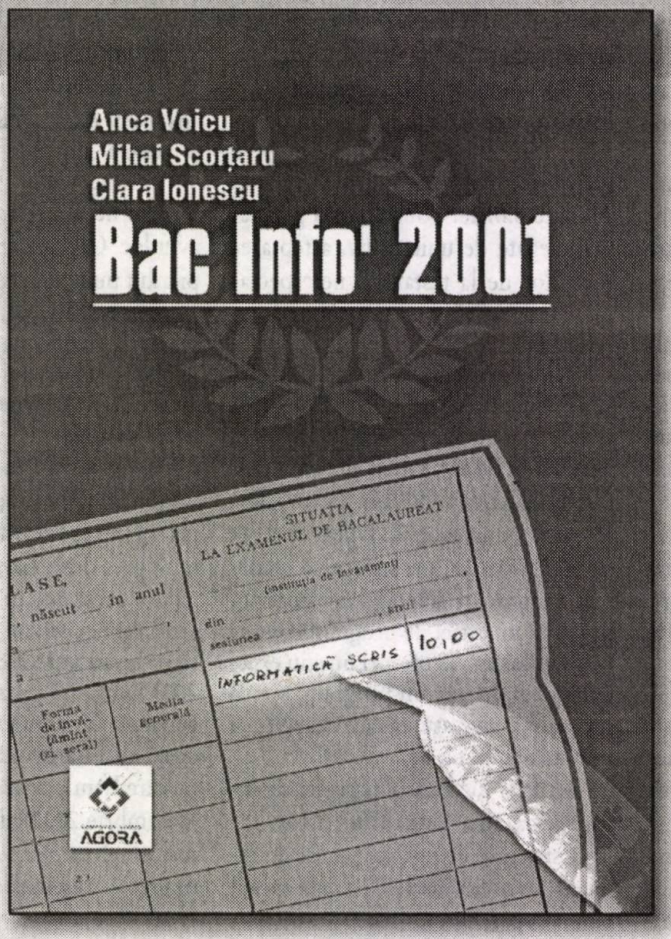
Și acum, urmează olimpiada națională de la *Focșani*. Să sperăm că nu se va întâmpla la fel ca anul trecut... Oricum, îi sfătuiesc pe cei care participă să nu depună contestații din principiu. Nu are nici un rost să-i "chinuiți" pe membrii comisiei.

Totuși, dacă credeți că ați fost nedreptățiți, nu ezitați! Dacă s-a greșit, greșeala trebuie reparată și nu poate fi reparată decât acolo. Dacă peste șase luni apare în *GInfo* un articol din care rezultă că ați fi meritat mult mai multe puncte, atunci nu se va rezolva nimic!

Mihai Scorțaru



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Faza JUDEȚEANĂ a olimpiadei de INFORMATICĂ pentru GIMNAZIU

Vă prezentăm în continuare enunțurile celor opt probleme propuse în acest an la faza județeană a olimpiadei de informatică pentru clasele V-VIII. Subiectele au fost aceleași în toate județele țării.

Clasa a V-a

P040301: Exponent

Se dă un număr natural n și o cifră k din mulțimea {2, 3, 5, 7}. Se cere să se afișeze exponentul lui k în descompunerea în factori primi a produsului $1 \cdot 2 \cdot 3 \cdot ... \cdot n$.

Date de intrare
De la tastatură se citesc numerele n și k.

Date de ieșire
Pe ecran se va afișa un singur număr natural care reprezintă exponentul cerut de problemă.

Restricție
- $1 \le n \le 100$.

Exemplu

Intrare	*Ieșire*
n=6	2
k=3	

Explicație
Se va afișa cifra 2 pentru că $1 \cdot 2 \cdot 3 \cdot ... \cdot 6 = 2^4 \cdot 3^2 \cdot 5$, deci 3 are exponentul 2 în descompunerea în factori primi a produsului $1 \cdot 2 \cdot 3 \cdot ... \cdot 6$.

Timp de execuție: 1 secundă/test

P040302: Pinocchio

În fiecare zi nelucrătoare din săptămână, *Pinocchio* spune câte o minciună datorită căreia nasul acestuia crește cu câte p centimetri pe zi.

Sâmbăta și duminica, când vine bunicul *Gepetto* acasă, pentru a nu-l supăra prea tare, *Pinocchio* reușește să nu spună nici o minciună, ba chiar uitându-se în oglindă observă că în fiecare din aceste zile lungimea nasului său scade cu câte un centimetru pe zi. Când începe o nouă săptămână, rămânând singur acasă, *Pinocchio* continuă șirul minciunilor.

Care este dimensiunea nasului lui *Pinocchio* după k zile, știind că inițial nasul său măsura n centimetri?

Date de intrare
De la tastatură se citesc valorile n, p și k.

Date de ieșire
Pe ecran se va afișa un singur număr natural, adică numărul de centimetri cerut de problemă.

Restricție
- $1 \le n \le 1000$, $1 \le k \le 256$, $1 \le p \le 100$.

Exemplu

Intrare	*Ieșire*
n=2	6
p=1	
k=8	

Timp de execuție: 1 secundă/test

Clasa a VI-a

P040303: Visul

Sinbad Marinarul visează că se află într-o peșteră cu comori. Peste tot se aflau cufere pline cu bijuterii și monede din aur, iar peștera era luminată de strălucirea lor. În timp

ce *Sinbad* se minuna de toate splendorile din jurul său, se auzi o voce misterioasă, care spuse:

"Există o posibilitate de a ajunge aici și, dacă reușești, toate aceste comori vor fi ale tale. Această peșteră se află în vârful muntelui Ararat, dar pe drum, duhurile rele vor încerca să te oprească. Tu trebuie să lupți cu ele și să le învingi în luptă dreaptă. Pentru a deschide peștera, trebuie să-ți amintești câte duhuri ai învins și să rostești cu voce tare formula magică. Formula magică este cel mai mic număr care are atât prima cifră, cât și numărul de cifre egale cu numărul duhurilor rele învinse de tine. Dar atenție! În plus, acest număr trebuie să aibă proprietatea că orice secvență de două cifre consecutive trebuie să fie numere prime diferite."

În acest moment, *Sinbad* s-a trezit și vrea să plece în căutarea comorii. Nu se teme de duhurile rele, dar știe câte calcule necesită căutarea formulei magice (și mai știe că la matematică nu se descurcă foarte bine). De aceea, vă roagă să-l ajutați. *Sinbad* vă va spune numărul n (reprezentând numărul duhurilor rele învinse de el), iar voi va trebui să-i spuneți formula magică. În cazul în care nu există un astfel de număr, spuneți-i lui *Sinbad* că a fost doar un vis și că e timpul să se pregătească pentru școală.

Date de intrare
De la tastatură se citește numărul n.

Date de ieșire
Pe ecran se va afișa un singur număr natural, adică cel cerut de problemă. Dacă nu există soluție, se va afișa mesajul "Nu exista".

Restricție
- $1 \le n \le 10$.

Exemplu

Intrare	*Ieșire*
n=3	311

Timp de execuție: 1 secundă/test

P040304: Gardul

Doi copii vopsesc un gard alcătuit din n scânduri pe care le vom numerota de la 1 la n astfel: primul ia o cutie de vopsea roșie cu care vopsește scândurile cu numărul p, $2 \cdot p$, $3 \cdot p$ etc. Al doilea procedează la fel, începe de la același capăt al gardului, dar ia o cutie de vopsea albastră și vopsește din q în q scânduri. Astfel, când vor termina de vopsit, gardul va avea multe scânduri nevopsite, unele scânduri vopsite în roșu, altele în albastru, iar altele în violet (cele care au fost vopsite și cu roșu și cu albastru).

Cunoscând numerele n, p și q calculați:
a) câte scânduri rămân nevopsite;
b) câte scânduri sunt vopsite în roșu;
c) câte scânduri sunt vopsite în albastru;
d) câte scânduri sunt vopsite în violet.

Date de intrare
De la tastatură se citesc n, p și q.

Date de ieșire
Pe ecran se vor afișa cele patru numere naturale corespunzătoare rezultatelor obținute la punctele a), b), c) și d) câte unul pe un rând ca în exemplul următor.

Restricție
- $1 \le n \le 100000$.

Exemplu

Intrare	*Ieșire*
n=25 p=4 q=6	a) 17 b) 4 c) 2 d) 2

Explicație
Exemplul corespunde situației următoare:

.	.	.	R	.	A	.	R	.	.	.	V	.	.	.	R	.	A	.	R	.	.	.	V	.

unde: A reprezintă albastru, R roșu și V violet.

Timp de execuție: 1 secundă/test

Clasa a VII-a

P040305: Șir

Se consideră șirul: a, b, ba, bab, babba, babbabab, ... Scrieți un program care să determine cel de-al n-lea termen al șirului.

Date de intrare
Fișierul de intrare **SIR.IN** conține o singură linie pe care se află numărul natural n.

Date de ieșire
Fișierul de ieșire **SIR.OUT** va conține o singură linie pe care se va afla al n-lea termen din șir.

Restricție
- $1 \le M \le 20$.

Exemplu

SIR.IN	**SIR.OUT**
4	bab

Timp de execuție: 1 secundă/test

P040306: Paranteze

Considerăm n șiruri formate din paranteze de două tipuri: paranteze rotunde și paranteze drepte. Parantezele se codifică în felul următor: paranteza rotundă deschisă cu 0, paranteza rotundă închisă cu 1, paranteza dreaptă deschisă cu 2, paranteza dreaptă închisă cu 3. Spre deosebire de convenția uzuală din matematică, aici pot exista paranteze ro-

tunde incluse în paranteze drepte, cât și paranteze drepte incluse în paranteze rotunde. Nu putem asocia unei paranteze rotunde deschise o paranteză dreaptă închisă sau invers.

Să se decidă dacă un astfel de șir este corect construit, în sensul că putem asocia corect două câte două parantezele de același tip.

Date de intrare

Fișierul text **PAR.IN** conține pe prima linie numărul n (numărul de șiruri ale testului). Apoi, pe fiecare dintre următoarele n linii se află un șir de numere L c_1 c_2 ... c_L separate printr-un singur spațiu. Pentru fiecare șir, numărul natural L reprezintă lungimea unui șir de paranteze codificat conform enunțului. Valorile c_1, c_2, ..., c_L reprezintă codurile parantezelor respective.

Date de ieșire

În fișierului text **PAR.OUT** se vor scrie n linii. Pe fiecare linie se va scrie unul dintre mesajele "Da", respectiv "Nu", reprezentând rezultatele verificării corectitudinii șirurilor. Ordinea lor va corespunde ordinii șirurilor din fișierul de intrare.

Restricție

- $1 \le n \le 10$, $1 \le L \le 500$.

Exemplu

PAR.IN	PAR.OUT
8	Da
6 0 2 3 1 0 1	Da
6 2 0 1 0 1 3	Nu
4 0 2 1 3	Nu
6 2 0 0 3 1 1	Nu
6 2 2 0 1 1 3	Nu
6 2 0 0 1 3 0	Da
10 2 2 0 1 0 1 3 3 0 1	Da
10 0 0 0 1 1 1 0 2 3 1	

Timp de execuție: 1 secundă/test

Clasa a VIII-a

P040307: Templu

Copa ajunge în *Orintia* unde există un templu cu mai multe niveluri, având baza un pătrat cu latura de lungime L. Primul nivel are înălțimea egală cu N, iar celelalte niveluri au înălțimea mai mare cu o unitate față de cel anterior. De exemplu, pentru $L = 5$ și $N = 3$, templul este următorul (imagini din avion, respectiv de la sol):

```
3 3 3 3 3
3 4 4 4 3        5
3 4 5 4 3      4 4 4
3 4 4 4 3    3 3 3 3 3
3 3 3 3 3
```

Copa deschide un document vechi și citește: "*Ca să afli cât aur este în templu, trebuie să însumezi numărul de metri de pe fiecare orizontală...*". Și *Copa* socotește: 3 + 3 + 3 + 3 + 3 = 15; 3 + 4 + 4 + 4 + 3 = 18; 3 + 4 + 5 + 4 + 3 = 19; celelalte fiind 18, respectiv 15. "*Apoi, trebuie să afli suma numerelor obținute...*". Iar *Copa* își notă numărul 85. "*Toate numerele obținute se lipesc pentru a forma cel mai mic număr posibil...*". Și *Copa* obține numărul: 151518181985. "*Din numărul acesta se caută cel mai mare număr de două cifre alăturate. Aceasta este cantitatea de aur din templu.*" Și *Copa* țipă de bucurie: 98!

Plecați în *Orintia*! Veți primi cele două numere N și L și vi se cere să determinați numărul obținut din sume și cantitatea de aur.

Date de intrare

Fișierul **TEMPLU.IN** conține pe prima linie numerele N și L separate printr-un singur spațiu.

Date de ieșire

Fișierul **TEMPLU.OUT** va conține două linii. Pe prima linie se va afla numărul obținut din sume, iar pe cea de-a doua linie cantitatea de aur.

Restricție

- $2 \le L \le 10$, $1 \le N \le 50$.

Exemplu

TEMPLU.IN	TEMPLU.OUT
3 5	151518181985
	98

Timp de execuție: 1 secundă/test

P040308: Tort

Un tort dreptunghiular de dimensiuni $M \times N$ trebuie împărțit în porții pătrate de aceeași mărime.

Găsiți numărul minim de porții care se pot obține și dimensiunea L a acestora. Atât dimensiunile dreptunghiului cât și ale pătratelor în care se împarte tortul sunt numere întregi.

Date de intrare

Fișierul **TORT.IN** va conține pe prima linie numerele M și N separate printr-un singur spațiu.

Date de ieșire

Fișierul **TORT.OUT** va conține, pe o singură linie, două numere naturale separate printr-un singur spațiu, primul reprezenând numărul minim de porții, iar celălalt dimensiunea L.

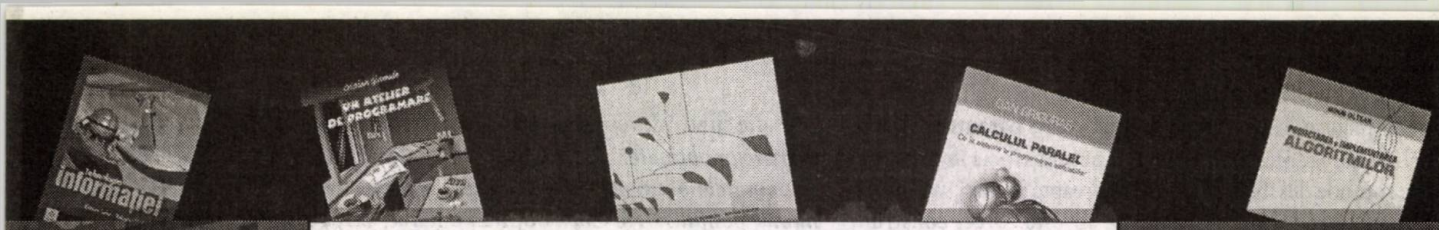
Restricție

- $1 \le M, N \le 10000$.

Exemplu

TORT.IN	TORT.OUT
20 24	30 4

Timp de execuție: 1 secundă/test



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

23 martie 2003

Faza JUDEȚEANĂ a olimpiadei de INFORMATICĂ pentru LICEE

La fel ca anul trecut, subiectele de la faza județeană a olimpiadei de informatică au fost aceleași în întreaga țară. Vă prezentăm în continuare enunțurile celor șase probleme propuse spre rezolvare.

Clasa a IX-a

P040309: Text

Vasile lucrează intens la un editor de texte. Un text este format din unul sau mai multe paragrafe. Orice paragraf se termină cu `Enter` și oricare două cuvinte consecutive din același paragraf sunt separate prin spații (unul sau mai multe). În funcție de modul de setare a paginii, numărul maxim de caractere care încap în pagină pe o linie este unic determinat (*Max*).

Funcția pe care *Vasile* trebuie să o implementeze acum este alinierea în pagină a fiecărui paragraf din text la stânga și la dreapta. Pentru aceasta el va trebui să împartă fiecare paragraf în linii separate de lungime *Max* (fiecare linie terminată cu `Enter`). Împărțirea se realizează punând numărul maxim posibil de cuvinte pe fiecare linie, fără împărțirea cuvintelor în silabe.

Pentru aliniere stânga-dreapta, el trebuie să repartizeze spații în mod uniform între cuvintele de pe fiecare linie, astfel încât ultimul caracter de pe linie să fie diferit de spațiu, iar numărul total de caractere de pe linie să fie egal cu *Max*. Excepție face numai ultima linie din paragraf, care rămâne aliniată la stânga (cuvintele fiind separate printr-un singur spațiu, chiar dacă linia nu este plină).

În general, este puțin probabil ca alinierea să fie realizabilă prin plasarea aceluiași număr de spații între oricare două cuvinte consecutive de pe linie. *Vasile* consideră că este mai elegant ca, dacă între unele cuvinte consecutive trebuie plasat un spațiu în plus față de alte perechi de cuvinte consecutive, acestea să fie plasate la începutul liniei.

Scrieți un program care să citească lungimea unei linii și textul dat și care să alinieze textul la stânga și la dreapta.

Date de intrare

Fișierul de intrare **`TEXT.IN`** conține pe prima linie *Max*, lungimea maximă a unui rând. Pe fiecare dintre următoarele linii se află câte un paragraf al textului care trebuie aliniat stânga-dreapta.

Date de ieșire

Fișierul de ieșire **`TEXT.OUT`** va conține textul aliniat stânga-dreapta.

Restricții și precizări

- $2 \leq Max \leq 1000$;
- lungimea maximă a unui cuvânt din text este de 25 de caractere și nu depășește *Max*;
- lungimea unui paragraf nu poate depăși 1000 de caractere;
- soluția este unică.

Exemple

`TEXT.IN`

```
20
Vasile are multe bomboane bune.
```

`TEXT.OUT`

```
Vasile   are   multe
bomboane bune.
```

`TEXT.IN`

```
20
Ana are mere.
Ion are multe pere galbene?
```

TEXT.OUT

```
Ana are mere.
Ion  are  multe pere
galbene?
```

Explicații

În cazul primului exemplu, pe prima linie au fost plasate câte trei spații între cuvintele consecutive.

În al doilea exemplu, paragraful "Ana are mere." (care are lungimea mai mică decât 20) a rămas aliniat la stânga, iar ultima linie din fiecare paragraf rămâne aliniată la stânga, cuvintele consecutive fiind separate printr-un singur spațiu. În al doilea paragraf, între "Ion" și "are" există două spații, între "are" și "multe" există două spații, iar între "multe" și "pere" un spațiu.

Timp de execuție: 1 secundă/test

P040310: Numere

Gigel este un mare pasionat al cifrelor. Orice moment liber și-l petrece jucându-se cu numere. Jucându-se astfel, într-o zi a scris pe hârtie 10 numere distincte de câte două cifre și a observat că printre acestea există două submulțimi disjuncte de sumă egală.

Desigur, Gigel a crezut că este o întâmplare și a scris alte 10 numere distincte de câte două cifre și spre surpriza lui, după un timp a găsit din nou două submulțimi disjuncte de sumă egală.

Dându-se 10 numere distincte formate din câte două cifre fiecare, determinați numărul de perechi de submulțimi disjuncte de sumă egală care se pot forma cu numere din cele date, precum și una dintre aceste perechi pentru care suma numerelor din fiecare dintre cele două submulțimi este maximă.

Date de intrare

Fișierul de intrare **NUMERE.IN** conține o singură linie pe care se află 10 numere naturale distincte, x_1, x_2, ..., x_{10}, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **NUMERE.OUT** va conține trei linii. Pe prima linie se va afla numărul de perechi de submulțimi de sumă egală, precum și suma maximă obținută, separate printr-un spațiu.

Pe linia a doua se vor afla elementele primei submulțimi, iar pe cea de-a treia linie se vor afla elementele celei de a doua submulțimi, separate prin câte un spațiu.

Restricții și precizări

- $10 \le x_i \le 99$, $\forall\ 1 \le i \le 10$;
- fiecare submulțime va avea cel puțin un element și cel mult 9;
- ordinea submulțimilor în perechi nu contează;
- perechea de submulțimi determinată nu este întotdeauna unică.

Exemplu

NUMERE.IN

```
60 49 86 78 23 97 69 71 32 10
```

NUMERE.OUT

```
130 276
78 97 69 32
60 49 86 71 10
```

Explicație

Pentru fișierul de intrare de la exemplul anterior, există 130 de soluții, suma maximă este 276. Din cele 10 numere date s-au folosit 9, prima submulțime având patru elemente, iar cea de-a doua cinci elemente.

Timp de execuție: 1 secundă/test

Clasa a X-a

P040311: Spirala

Se consideră un automat de criptare format dintr-un tablou bidimensional cu n linii și n coloane. Tabloul conține toate numerele cuprinse între 1 și n^2, așezate "șerpuit" pe linii, de la prima la ultima linie, pe liniile impare pornind de la stânga spre dreapta, iar pe cele pare de la dreapta spre stânga (ca în figura 1).

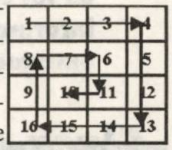


Figura 1

Numim "amestecare" operația de desfășurare în spirală a valorilor din tablou în ordinea indicată de săgeți și de reașezare a acestora în același tablou, "șerpuit" pe linii, ca și în cazul precedent.

1	2	3	4
14	13	12	5
15	16	9	8
10	11	6	7

Figura 2

De exemplu, desfășurarea tabloului duce la obținerea șirului: 1 2 3 4 5 12 13 14 15 16 9 8 7 6 11 10, iar reașezarea acestuia în tablou conduce la obținerea unui nou tablou reprezentat în figura 2.

După orice operație de amestecare se poate relua procedeul, efectuând o nouă amestecare. S-a observat un fapt interesant: că după un număr de amestecări, unele valori ajung din nou în poziția inițială (pe care o aveau în tabloul de pornire).

De exemplu, după două amestecări, tabloul de dimensiuni 4 × 4 conține 9 dintre elementele sale în exact aceeași poziție în care se aflau inițial (vezi elementele marcate din figura 3).

1	2	3	4
6	7	8	5
11	10	15	14
16	9	12	13

Figura 3

Dându-se numerele naturale n și k, scrieți un program care să determine numărul minim de amestecări, ale unui tablou cu n linii și n coloane, necesare pentru a ajunge la un tablou cu exact k elemente aflate din nou în poziția inițială.

Date de intrare

Fișierul de intrare **SPIRALA.IN** conține pe prima linie cele două numere n și k despărțite printr-un singur spațiu.

Date de ieșire

Fișierul de ieșire **SPIRALA.OUT** va conține o singură linie pe care se va afla numărul de amestecări determinat.

Restricții și precizări

- $3 \leq n \leq 50$;
- numărul minim de amestecări necesare nu va depăși valoarea $2 \cdot 10^9$.

Exemple

SPIRALA.IN

```
4 9
```

SPIRALA.OUT

```
2
```

SPIRALA.IN

```
6 36
```

SPIRALA.OUT

```
330
```

Timp de execuție: 1 secundă/test

P040312: Taxe

Într-o țară în care corupția este în floare și economia la pământ, pentru a obține toate aprobările necesare în scopul demarării unei afaceri, investitorul trebuie să treacă prin mai multe camere ale unei clădiri în care se află birouri.

Clădirea are un singur nivel în care birourile sunt lipite unele de altele, formând un caroiaj pătrat de dimensiune $n \times n$. Pentru a facilita accesul în birouri, toate camerele vecine au uși între ele. În fiecare birou se află un funcționar care pretinde o taxă de trecere prin cameră (taxă ce poate fi, pentru unele camere, egală cu 0). Investitorul intră încrezător prin colțul din stânga-sus al clădirii (cum se vede de sus planul clădirii) și dorește să ajungă în colțul opus al clădirii, unde este ieșirea, plătind o taxă totală cât mai mică.

Știind că el are în buzunar S *euro* și că fiecare funcționar îi ia taxa de cum intră în birou, se cere să se determine dacă el poate primi aprobările necesare și, în caz afirmativ, care este suma maximă de bani care îi rămâne în buzunar la ieșirea din clădire.

Date de intrare

Fișierul de intrare **TAXE.IN** conține pe prima linie cele două numere S și n despărțite printr-un spațiu, iar pe următoarele n linii, câte n numere separate prin spații care reprezintă taxele cerute de funcționarii din fiecare birou.

Date de ieșire

Fișierul de ieșire **TAXE.OUT** conține o singură linie pe care se află numărul maxim de *euro* care îi rămân în buzunar sau valoarea -1, dacă investitorului nu-i ajung banii pentru a obține aprobarea.

Restricții și precizări

- $3 \leq n \leq 100$;
- $1 \leq S \leq 10000$;
- valorile reprezentând taxele cerute de funcționarii din birouri sunt numere naturale, o taxă nedepășind valoarea de 200 de *euro*.

Exemplu

TAXE.IN

```
10 3
1 2 5
1 3 1
0 8 1
```

TAXE.OUT

```
3
```

Timp de execuție: 1 secundă/test

Clasele a XI-a și a XII-a

P040313: Compus

La ultima expediție pe *Marte* a fost descoperit un compus organic necunoscut. Acest compus este acum studiat în laboratoarele *NASA*.

Cercetătorii au descoperit că acest compus este format numai din atomi de *hidrigen* (H), *ixigen* (I) și *carbin* (C) și are masa moleculară M.

Se știe că regulile de formare a compușilor organici pe *Marte* sunt următoarele:

- un atom de *carbin* se poate lega de oricare dintre atomii de C, H și I cu oricâte dintre cele 4 legături pe care le are (astfel, în combinația **H – C = C** primul atom de *carbin* se leagă prin două legături de alt atom de *carbin* și cu o legătură de alt atom de *hidrigen*);
- un atom de *hidrigen* se poate lega numai de un atom de *carbin* cu singura legătură pe care o posedă;
- un atom de *ixigen* se poate lega numai de atomi de *carbin* cu cele două legături pe care le posedă;
- un compus este un ansamblu cu proprietatea că toți atomii de *carbin* sunt legați conex între ei și nu există vreun atom cu una sau mai multe legături libere (nelegate de un alt atom).

Combinația **H – C = C** nu este un compus, deoarece atomii de *carbin* mai au legături libere.

Cercetătorii au în vedere studiul categoriilor de compuși, făcând distincție între doi compuși numai dacă aceștia diferă prin numărul de atomi de *carbin*, de *ixigen* sau de *hidrigen*.

Scrieți un program care să determine câți compuși distincți formați din atomi de *carbin*, *hidrigen* și *ixigen* (cel puțin unul din fiecare) și care au masa moleculară M există.

Date de intrare

Fișierul de intrare **COMPUS.IN** conține pe prima linie masa moleculară a compusului.

Date de ieşire

Fişierul de ieşire **COMPUS.OUT** va conţine o singură linie pe care se va afla numărul de compuşi determinat.

Restricţii şi precizări

- $30 \le M \le 1000000$;
- masa atomului de *H* este 1, masa atomului de *C* este 5, iar masa atomului de *I* este 3. Masa moleculară a unui compus este egală cu suma maselor atomilor din care este constituit compusul respectiv;
- ordinea în care sunt "utilizate" legăturile unui atom nu contează. De asemenea, nici ordinea atomilor sau legăturile interne dintre ei nu contează, atâta timp cât respectă regulile de formare enunţate.

Observaţii

Există un singur compus cu masa moleculară 10: cel format cu un atom de *C*, doi atomi de *H* şi un atom de *I* ($5 + 2 \cdot 1 + 3 = 10$), compus ale cărui legături pot fi reprezentate astfel:

```
H – C = C
    |
    H
```

Se pot obţine trei compuşi cu masa moleculară 40: (5*C*, 6*H*, 3*I*), (6*C*, 4*H*, 2*I*) şi (7*C*, 2*H*, 1*I*) care sunt prezentaţi în figura 4.

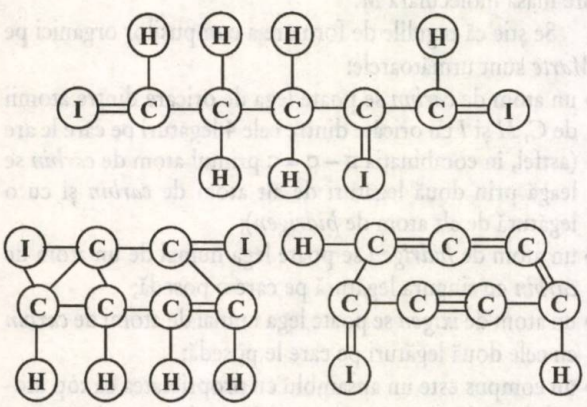


Figura 4

Reprezentarea cu legături a oricăruia dintre compuşi nu este unică. Orice altă combinaţie corespunzătoare aceluiaşi triplet nu este considerată ca fiind un compus distinct.

Exemple

COMPUS.IN
```
40
```

COMPUS.OUT
```
3
```

COMPUS.IN
```
120
```

COMPUS.OUT
```
28
```

Timp de execuţie: 1 secundă/test

P040314: Zmeu

Un *zmeu* cu *n* capete călătoreşte din poveste în poveste, iar în poveştile tradiţionale întâlneşte câte un *Făt-Frumos* care-l mai retează de câteva capete, în timp ce în poveştile moderne salvează omenirea mâncând în timp record, cu toate capetele lui, insecte ucigaşe apărute prin mutaţii genetice. Într-o seară, el îşi planifică o succesiune de poveşti cărora să le dea viaţă. El ştie *p* poveşti numerotate de la 1 la *p*, durata fiecăreia şi numărul de capete pe care le pierde în fiecare poveste. Mai ştie o mulţime de *k* perechi de poveşti, semnificând faptul că a doua poveste din pereche nu poate fi spusă după prima poveste din pereche.

Ştiind că trebuie să înceapă cu povestea 1 şi să încheie succesiunea cu povestea *p*, ajutaţi bietul zmeu să aleagă una sau mai multe poveşti intermediare, astfel încât durata totală să fie minimă şi să rămână cu cel puţin un cap la sfârşitul tuturor poveştilor.

Date de intrare

Fişierul de intrare **ZMEU.IN** conţine pe prima linie numerele *n*, *p* şi *k* despărţite prin câte un spaţiu. Pe fiecare din următoarele *p* linii se află câte o pereche de numere d_i şi c_i (separate prin câte un spaţiu) care reprezintă durata şi numărul de capete tăiate pentru fiecare poveste. Iar pe ultimele *k* linii se află câte o pereche de numere p_i şi p_j (separate prin câte un spaţiu) care semnifică faptul că povestea p_j nu poate fi spusă după povestea p_i.

Date de ieşire

Fişierul de ieşire **ZMEU.OUT** va conţine o singură linie pe care se va afla un număr natural care reprezintă durata minimă a succesiunii de poveşti sau valoarea -1 dacă nu există o astfel de succesiune.

Restricţii şi precizări

- $2 \le n \le 500$;
- $1 \le n \le 200$;
- $1 \le n \le 30000$;
- valorile reprezentând duratele şi numărul de capete sunt numere naturale (duratele fiind strict pozitive), nedepăşind valoarea 10.

Exemplu

ZMEU.IN
```
10 4 2
2 6
4 0
1 3
3 3
3 2
4 3
```

ZMEU.OUT
```
9
```

Timp de execuţie: 1 secundă/test

Rundele #13 - #16

Vom prezenta în cadrul acestui articol câteva informații referitoare la desfășurarea a patru runde ale concursului de programare organizat de revista noastră. De asemenea, veți putea vedea clasamentul primilor o sută clasați după desfășurarea primelor 16 runde.

P030311: Grupuri

Această problemă este echivalentă cu problema *Soldați* propusă spre rezolvare la faza locală a *Olimpiadei de Informatică* din județul *Cluj*. Rezolvarea ei a fost prezentată în numărul anterior (*GInfo* 13/3 - martie 2003) al revistei noastre.

Pentru această problemă am primit 32 de soluții pentru care a fost acordat punctajul maxim. Concurenții care au primit 100 de puncte la această etapă a concursului nostru sunt: *Adrian Cârcu (Bistrița), Adrian Nasui (Slobozia), Alexandru-Cosmin Vlajoaga (Pitești), Alin Coman (Târgu Jiu), Andrei Csibi (Cluj-Napoca), Andrei-Gabriel Ion (Constanța), Andrei Gönczi (Arad), Andrei Homescu (Târgu Jiu), Aurelian Marinescu (Brașov), Bogdan Nicolae (București), Ciprian Cană (Vaslui), Constantin Jucovschi (Chișinău - Republica Moldova), Constantin Răuțu (Botoșani), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Daniel Nechifor (Vaslui), Dan-Ionuț Fechete (Suceava), Dumitru Codreanu (Chișinău - Republica Moldova), Gabriel-Alexandru Rudeanu (Iași), Iulian Blajut (Bacău), Iuliu Konya (Baia Mare), Ivan Kazmenko (Sankt Petersburg - Federația Rusă), Lucian Suditu (Vaslui), Marius Airinei (Bacău), Marius Nicolae (Slatina), Maximilian Machedon (București), Mihai Stanimir (Bumbești-Jiu), Mircea Pasoi (Ploiești), Mugurel-Ionuț Andreica (București), Răzvan Albu (București), Silvestru-Cosmin Negrușeri (Bistrița)* și *Tiberiu Chiorean (Baia Mare).*

Numărul total al concurenților care au trimis soluții pentru această problemă a fost 50; așadar, aproape două treimi dintre ei au rezolvat-o corect.

P030312: Războiul stelelor

Problema se reduce la determinarea unei colorări a nodurilor unui graf, astfel încât oricare două noduri să fie colorate folosind culori cu numere de ordine diferite. În plus, suma tuturor numerelor de ordine ale culorilor trebuie să fie minimă.

Problema colorării nodurilor unui graf este *NP*-completă, motiv pentru care nu putea fi creat un program care să determine cea mai bună soluție în limita de timp impusă. Ca urmare, această rundă a fost un "concurs de euristici". Câștigătorul acestui miniconcurs a fost *Adrian Cârcu* din *Bistrița*, care a avut cele mai bune rezultate pentru 17 dintre cele 20 de teste.

Lista celor care au obținut cel puțin 90 dintre cele 100 de puncte care puteau fi obținute la această rundă este prezentată în continuare:

1. Adrian Cârcu, Bistrița	*100*
2. Ciprian Cană, Vaslui	*98*
2. Dan Ghinea, București	*98*
2. Maximilian Machedon, București	*98*
2. Mugurel-Ionuț Andreica, București	*98*
6. Csaba Pătcaș, Cluj-Napoca	*97*
6. Ivan Kazmenko, Federația Rusă	*97*
6. Ștefan Ciobâcă, Suceava	*97*
9. Silvestru-Cosmin Negrușeri, Bistrița	*96*
10. Andrei Blaj, Suceava	*95*
11. Constantin Răuțu, Botoșani	*94*
11. Dan-Ionuț Fechete, Suceava	*94*
13. Bogdan Nicolae, București	*93*
13. Dinh Ngoc Thang, Vietnam	*93*
15. Aurelian Marinescu, Brașov	*92*
15. Iuliu Konya, Baia Mare	*92*
17. Cătălin-Valeriu Liță, Bârlad	*91*
17. Daniel Nechifor, Vaslui	*91*
17. Dumitru Codreanu, Republica Moldova	*91*
17. Lucian Suditu, Vaslui	*91*
17. Marius Nicolae, Slatina	*91*
17. Mihai Stanimir, Bumbești-Jiu	*91*
17. Mihail Popa, București	*91*
17. Răzvan Albu, București	*91*
25. Adrian Nasui, Slobozia	*90*
25. Andrei Gönczi, Arad	*90*
25. Cristian Bojinovici, București	*90*
25. Cristian Cernat, Târgu Jiu	*90*

La această rundă au fost evaluate soluțiile trimise de 48 dintre concurenții înscriși la această a patra ediție a concursului de programare *Bursele Agora*.

P030313: Galaxie

Problema se reduce la determinarea numărului de cicluri din care face parte fiecare muchie a unui graf neorientat. Problema este nedeterminist polinomială deoarece numărul acestor cicluri poate crește exponențial în funcție de numărul muchiilor (sau al nodurilor) din graf.

Fișierele de ieșire pentru această problemă au fost "deschise". Până la urmă doar șapte dintre cele douăzeci de fișiere au fost luate în considerare în cadrul evaluării.

Datorită acestui fapt, un număr foarte mare de concurenți au obținut punctajul maxim. Aceștia sunt: *Adrian Cârcu (Bistrița), Andrei Homescu (Târgu Jiu), Bogdan Nicolae (București), Călin Bindea (Bistrița), Ciprian Cană (Vaslui), Constantin Răuțu (Botoșani), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Daniel Nechifor (Vaslui), Eugen Sorbalo (Chișinău - Republica Moldova), Ionuț-Bogdan Petre (Plopeni), Iulian Blajut (Bacău), Iuliu Konya (Baia Mare), Ivan Kazmenko (Sankt Petersburg - Federația Rusă), Lucian Suditu (Vaslui), Maximilian Machedon (București), Mihai Stanimir (Bumbești-Jiu), Mugurel-Ionuț Andreica (București), Paul-Horațiu Stan (Cluj-Napoca), Răzvan Albu (București), Silvestru-Cosmin Negrușeri (Bistrița), Ștefan Ciobâcă (Suceava)* și *Tiberiu Chiorean (Baia Mare).*

Numărul concurenților care au trimis fișiere de ieșire pentru această problemă a fost 39.

P010314: Săgeți

Cercurile de pe perete pot fi privite ca fiind nodurile unui graf orientat ale cărui arce sunt date de săgețile dintre cercuri; sensurile arcelor sunt date de sensurile săgeților.

În aceste condiții, problema se reduce la determinarea unui nod al grafului care are gradul interior $N - 1$ (la el ajung arce de la toate celelalte noduri) și gradul exterior 0 (nu există nici un arc care pleacă de la nodul respectiv).

Datorită limitei de $2 \cdot N$ interogări (apeluri ale funcției `IsArrow()`) pe care le avem la dispoziție va trebui să găsim un algoritm eficient de determinare a unui astfel de nod.

Inițial, vom presupune că nodul căutat este identificat prin 1. Vom verifica respectarea condițiilor parcurgând celelalte noduri. Atât timp cât există o săgeată de la nodul curent la nodul 1, vom continua parcurgerea.

Să presupunem că identificăm un nod i pentru care condiția nu mai este satisfăcută. În această situație putem presupune că nici unul dintre nodurile cuprinse între 2 și $i - 1$ nu reprezintă soluția, deoarece de la ele pleacă arce spre nodul 1.

Evident, nici nodul 1 nu este soluția căutată, deoarece nu există o săgeată de la el la nodul i. Ca urmare, problema se reduce la determinarea unei soluții pentru subgraful format din nodurile cu numere de ordine cuprinse între i și N.

Vom aplica acest algoritm până în momentul în care ajungem să luăm în considerare și nodul N. În acest moment soluția va fi dată fie de "candidatul" curent, în cazul în care se respectă condițiile pentru acel nod și nodul N, fie de nodul N, în caz contrar. În final, soluția obținută este furnizată ca rezultat.

Trebuie observat faptul că enunțul ne asigură că soluția există și este unică. Deoarece parcurgem $N - 1$ noduri și la fiecare pas efectuăm o interogare, numărul total al interogărilor va fi întotdeauna $N - 1$, deci condiția impusă asupra numărului maxim de interogări este respectată.

Așadar, limita de $2 \cdot N$ interogări impusă în enunț este arhisuficientă.

Problema s-a dovedit foarte simplă, 43 dintre cei 54 de concurenți care au trimis soluții au obținut punctajul maxim. Aceștia sunt: *Adrian Cârcu (Bistrița), Alex Florescu (București), Alexandru-Cosmin Vlajoaga (Pitești), Alin Coman (Târgu Jiu), Andrei Blaj (Suceava), Andrei Gönczi (Arad), Asif-ul Haque (Dhaka - Bangladej), Aurelian Marinescu (Brașov), Bogdan Nicolae (București), Cătălin-Valeriu Liță (Bârlad), Ciprian Cană (Vaslui), Constantin Răuțu (Botoșani), Cosmin Molea (Bascov), Cristian Bojinovici (București), Cristian Cernat (Târgu Jiu), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Daniel Nechifor (Vaslui), Dan-Ionuț Fechete (Suceava), Dumitru Codreanu (Chișinău - Republica Moldova), Eugen Sorbalo (Chișinău - Republica Moldova), Florin Antonescu (Băicoi), Gabriel-Alexandru Rudeanu (Iași), Ionuț-Bogdan Petre (Plopeni), Iulian Blajut (Bacău), Iuliu Konya (Baia Mare), Ivan Kazmenko (Sankt Petersburg - Federația Rusă), Lucian Suditu (Vaslui), Marius Airinei (Bacău), Marius-Dan Dragomir (Alba Iulia), Marius Nicolae (Slatina), Maximilian Machedon (București), Mugurel-Ionuț Andreica (București), Paul-Horațiu Stan (Cluj-Napoca), Răzvan Albu (București), Sergiu Spătaru (Zimandu Nou), Silvestru-Cosmin Negrușeri (Bistrița), Silviu Gânceanu (Bârlad), Silviu Julean (Arad), Slotta Johannes (Jena - Germania), Ștefan Ciobâcă (Suceava), Tiberiu Chiorean (Baia Mare)* și *Victor Podeanu (București).*

Despre Bursele Agora

Ediția din acest an a concursului de programare *Bursele Agora* se apropie de sfârșit. În continuare veți găsi lista primilor clasați după desfășurarea a 16 dintre cele 20 de runde, precum și enunțurile problemelor propuse spre rezolvare la ultimele patru etape.

Ne exprimăm speranța că acest concurs s-a dovedit util pentru toți cei care au trimis rezolvări și că rezultatele voastre la concursurile la care veți participa vor fi mai bune datorită experienței acumulate în cadrul concursului nostru.

În numărul următor vom prezenta detalii referitoare la rezolvarea problemelor de la ultimele patru runde, precum și un bilanț general referitor la această ediție.

Mulțumim tuturor celor care au participat la ediția 2002/2003 a concursului nostru și vă așteptăm și la următoarea ediție.

Vă urăm din nou mult succes la concursurile de programare la care veți participa!

Clasament după 16 runde

Adrian Cârcu, Bistrița	*1571*
Mugurel-Ionuț Andreica, București	*1501*
Ivan Kazmenko, Sankt Petersburg - Federația Rusă	*1425*
Silvestru-Cosmin Negrușeri, Bistrița	*1361*
Csaba Pătcaș, Cluj-Napoca	*1358*
Dan Ghinea, București	*1341*
Ciprian Cană, Vaslui	*1297*
Dumitru Codreanu, Chișinău - Republica Moldova	*1291*
Ștefan Ciobâcă, Suceava	*1285*
Bogdan Nicolae, București	*1265*

11. Aurelian Marinescu, Brașov	1221	41. Eugen Sorbalo, R. Moldova	427	71. Vlad Dascălu, Bacău	226
12. Daniel Nechifor, Vaslui	1142	42. AlexandruVlajoaga, Pitești	422	72. Răzvan Tigaeru, Călărași	220
13. Tiberiu Chiorean, Baia Mare	1135	43. Victor Costan, București	397	73. Călin Ciutu, București	214
14. Iulian Blajut, Bacău	1131	44. Andrei Gönczi, Arad	393	74. Horea Cărămizaru, Craiova	212
15. Iuliu Konya, Baia Mare	1130	45. Marian Raica, Drăgutești	374	75. Andrei Păduraru, Vaslui	200
16. M. Machedon, București	1080	46. M. Pit-Rada, Dr. Tr. Severin	364	75. Sorin Bălășioiu, Craiova	200
17. Dan-Ionuț Fechete, Suceava	1069	47. Ștefăniță Fechete, Suceava	353	77. Alexandru Constantin, Pitești	196
18. Ionuț-Bogdan Petre, Plopeni	1038	47. Cosmin Molea, Bascov	353	77. Cristian Cernat, Târgu Jiu	196
19. Andrei Homescu, Târgu Jiu	1024	49. Radu Ștefan, Brașov	352	79. S. Spătaru, Zimandu Nou	194
20. Lucian Suditu, Vaslui	1009	50. Mohamed Taha, Egipt	350	80. Alexandru Stan, Ploiești	193
21. Marius Nicolae, Slatina	975	51. Adrian Balașko, Zalău	349	81. Victor Podeanu, București	187
22. Răzvan Albu, București	953	52. Mihaela Marin, Codlea	348	82. Ralph Bottesch, Sibiu	166
23. Radu Vatavu, Suceava	875	53. Bogdan Pănuș, Satu Mare	334	83. Alexandra Constantin, Ploiești	164
24. Cornel Mihăilă, Brașov	837	54. Radu Berinde, București	324	84. Saratov Team, Rusia	155
25. Cristian Bojinovici, București	818	55. Mihai Stanimir, Bumbești-Jiu	316	84. Ștefan Bucur, București	155
26. Mihail Popa, București	802	56. Asif-ul Haque, Bangladej	311	86. Natalia Baranova, Rusia	154
27. Florin Antonescu, Băicoi	795	57. Ágota Turánicz, Cluj-Napoca	307	87. Alex Florescu, București	153
28. Alin Coman, Târgu Jiu	790	58. Lego Haryanto, S.U.A.	306	88. Ioan Stan, Aiud	151
29. Constantin Răuțu, Botoșani	770	59. Robert-Emanuel Nica, Deleni	305	89. A.G. Matei, Câmpulung M.	147
30. C-tin Jucovschi, R. Moldova	698	60. Andrei Csibi, Cluj-Napoca	300	90. Daniel Păsăilă, Suceava	144
31. Andrei-G. Ion, Constanța	673	60. Gabriel Rudeanu, Iași	300	91. Bogdan Chiriţoiu, Ploiești	143
32. Călin Bindea, Bistrița	656	62. Marius Dragomir, Alba Iulia	283	92. Vincent Groenhuis, Olanda	137
33. Slotta Johannes, Germania	599	63. Valentin Bran, Satu Mare	277	93. Alpár F. Perini, Sf. Gheorghe	131
34. Adrian Nasui, Slobozia	585	64. Alexandru Tomescu, București	260	94. Vasile Răcășan, Codlea	130
35. Mircea Pasoi, Ploiești	547	65. Alexandru Moșoi, Bacău	256	95. Miron Emilian, Bacău	125
36. Marius Airinei, Bacău	524	66. Paul H. Stan, Cluj-Napoca	251	96. Vlad Bălan, Bacău	120
37. Silviu Hagea, Călărași	518	67. Csaba Andras, Oradea	246	97. M.O. Dâmboiu, București	117
38. Marius-Dorin Moraru, Buftea	497	68. Cătălin-Valeriu Liță, Bârlad	245	98. Dragoș Scripniciuc, Timișoara	114
39. Andrei Blaj, Suceava	461	69. Costel-Cătălin Antohi, Tecuci	236	98. Ștefan Nour, Piatra Neamț	114
40. Alexandru Sofronia, Bacău	458	70. Angelo Lupuț, Satu Mare	235	100. Andrei Gheorghe, București	111

Bursele AGORA

Vă prezentăm în continuare enunțurile problemelor propuse spre rezolvare la ultimele patru runde ale celei de-a patra ediții a concursului de programare Bursele Agora.

P040315: Heroes of Might & Magic

Magicianul *Hexis* controlează un număr de N castele ale căror coordonate se cunosc. Nu există trei castele ale căror coordonate sunt puncte coliniare. Pentru a se proteja de iminentul atac al adversarilor săi, *Hexis* decide să trimită câte un erou în fiecare dintre orașele aflate pe perimetrul teritoriului. Perimetrul celor N castele este poligonul de perimetru minim care conține în interior sau în vârfuri toate castelele. Datorită faptului că *Hexis* știe că, mai devreme sau mai târziu, acest perimetru va fi străpuns, el organizează apărarea în continuare astfel:

- castelele aflate pe perimetru (în vârfurile poligonului) sunt considerate a fi pierdute;
- pentru castelele rămase se determină un nou perimetru și în castelele aflate pe noul perimetru sunt trimiși câte doi eroi;
- evident, și acest perimetru va fi străpuns, deci este organizat un al treilea nivel de apărare care va fi dat de perimetrul castelelor rămase; în castelele de pe acest perimetru vor fi trimiși câte trei eroi;
- procedeul va continua (numărul eroilor crește cu 1 pentru fiecare nou perimetru) până în momentul în care nu va mai rămâne nici un castel.

Va trebui să determinați, pentru fiecare castel în parte, numărul eroilor care vor fi trimiși în castelul respectiv.

Date de intrare

Prima linie a fișierului de intrare **HOMM.IN** conține numărul N al castelelor de pe teritoriul lui *Hexis*. Fiecare dintre următoarele N linii va conține o pereche de numere reale reprezentând coordonatele unui castel, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **HOMM.OUT** trebuie să conțină N linii; fiecare linie corespunde unui castel. O astfel de linie va conține un singur număr care reprezintă numărul eroilor trimiși în castelul respectiv.

Ordinea castelelor din fișierul de ieșire trebuie să respecte ordinea acestora din fișierul de intrare.

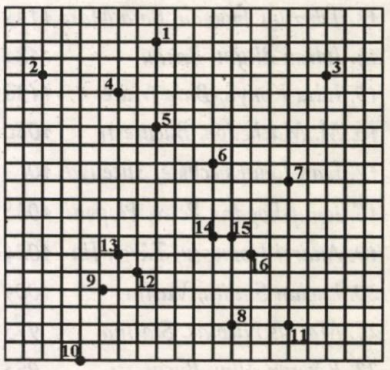
Restricții și precizări

- $1 \le N \le 1000$;
- coordonatele castelelor sunt numere reale cu cel mult trei zecimale exacte;
- ultimul perimetru poate conține unul sau două castele.

Exemplu

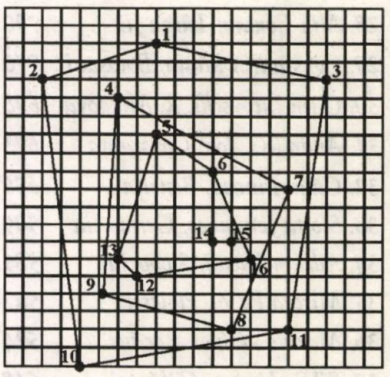
HOMM.IN

```
16
8 2
2 4
17 4
6 5
8 7
11 9
15 10
12 18
5.1 16
4 20
15 18
7 15
6 14
11 13
12 13
13 14
```



HOMM.OUT

```
1
1
1
2
3
3
2
2
2
1
1
3
3
4
4
3
```



Timp de execuție: 1 secundă/test

P040316: Fotbal

În *Liga de Fotbal Agora* participă N echipe, identificate prin numere cuprinse între 1 și N. Fiecare echipă joacă exact o partidă cu toate celelalte N - 1 echipe. Aveți la dispoziție un clasament final și va trebui să stabiliți rezultatele fiecărei partide, astfel încât să obțineți un clasament cât mai apropiat de cel dat. Pentru fiecare echipă se cunoaște numărul total al victoriilor, numărul total al partidelor terminate la egalitate și numărul total al înfrângerilor. Suma acestor trei numere va fi, pentru fiecare echipă, N - 1. De asemenea, pentru fiecare echipă se cunoaște numărul total al golurilor înscrise și numărul total al golurilor primite. Numărul total al golurilor înscrise de cele N echipe va fi egal cu numărul total al golurilor primite de cele N echipe.

Date de intrare

Prima linie a fișierului de intrare **FOOTBALL.IN** conține numărul N al echipelor participante. Fiecare dintre următoarele N linii conține câte cinci numere, separate prin câte un spațiu, reprezentând numărul victoriilor, al partidelor terminate la egalitate, al înfrângerilor, al golurilor marcate, respectiv al golurilor primite pentru o anumită echipă.

Date de ieșire

Datele de ieșire vor fi furnizate sub forma unei matrice pătratice de dimensiune N. Pentru fiecare pereche (i, j), $i < j$ elementul a_{ij} va conține numărul golurilor marcate de echipa i în partida dintre echipele i și j, iar elementul a_{ji} va conține numărul golurilor marcate de echipa j în partida dintre echipele i și j. Elementele de forma a_{ii} vor avea valoarea 0.

Fișierul de ieșire **FOOTBALL.OUT** va conține N linii pe care se vor afla câte N numere separate prin câte un spațiu. Aceste numere vor reprezenta elementele matricei descrise.

Restricții și precizări

- $2 \le N \le 1000$;
- ordinea echipelor din fișierul de ieșire trebuie să respecte ordinea din fișierul de intrare;
- într-o partidă o echipă poate marca cel mult zece goluri;
- întotdeauna va putea fi obținut un clasament identic cu cel dat.

Exemplu

FOOTBALL.IN

```
4
1 1 1 4 5
1 2 0 5 3
2 1 0 7 4
0 0 3 2 6
```

FOOTBALL.OUT

```
0 1 2 1
1 0 1 3
4 1 0 2
0 1 1 0
```

Modalitatea de acordare a punctajului

Vom considera că, pentru fiecare test, se vor putea obține cel mult X puncte. Pentru datele de ieșire furnizate de concurenți se va construi un clasament care va fi comparat cu cel dat. Pentru fiecare echipă se vor putea obține cel mult $P = X / N$ puncte.

Pentru o echipă, dacă numărul W' al victoriilor din clasamentul obținut este egal cu cel din clasamentul dat (W), se vor acorda $0.2 \cdot P$ puncte. În caz contrar, se vor acorda $\max(0, (1 - |W - W'| / (W + 1)) \cdot 0.2 \cdot P)$ puncte.

Dacă numărul D' al partidelor egale din clasamentul obținut este egal cu cel din clasamentul dat (D), se vor acorda $0.1 \cdot P$ puncte. În caz contrar, se vor acorda $\max(0, (1 - |D - D'| / (D + 1)) \cdot 0.1 \cdot P)$ puncte.

Dacă numărul L' al partidelor egale din clasamentul obținut este egal cu cel din clasamentul dat (L), se vor acorda $0.1 \cdot P$ puncte. În caz contrar, se vor acorda $\max(0, (1 - |L - L'| / (L + 1)) \cdot 0.1 \cdot P)$ puncte.

Dacă numărul S' al golurilor marcate din clasamentul obținut este egal cu cel din clasamentul dat (S), se vor acorda $0.3 \cdot P$ puncte. În caz contrar, se vor acorda $\max(0, (1 - |S - S'| / (S + 1)) \cdot 0.3 \cdot P)$ puncte.

Dacă numărul R' al golurilor primite din clasamentul obținut este egal cu cel din clasamentul dat (R), se vor acorda $0.3 \cdot P$ puncte. În caz contrar, se vor acorda $\max(0, (1 - |R - R'| / (R + 1)) \cdot 0.3 \cdot P)$ puncte.

Punctajul pentru un test va fi trunchiat la două zecimale exacte. Punctajul final va fi obținut prin adunarea punctajelor de la fiecare test și rotunjirea acestuia la cel mai apropiat număr întreg.

Să presupunem că, pentru exemplul dat, un concurent furnizează următorul fișier de ieșire:

FOOTBALL.OUT

```
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
```

Clasamentul obținut pe baza acestui fișier este:

```
0 3 0 3 3
0 3 0 3 3
0 3 0 3 3
0 3 0 3 3
```

Presupunem că pentru acest test pot fi obținute cel mult $X = 5$ puncte. Datorită faptului că avem patru echipe, pentru fiecare echipă vor fi acordate cel mult $P = 5 / 4 = 1.25$ puncte.

Pentru prima echipă avem:

- W = 1,W' = 0: ((1-|1-0|/(1+1))*0.2*1.25 = **0.125** puncte
- D = 1, D' = 3: ((1-|1-3|/(1+1))*0.1*1.25 = **0** puncte

- L = 1, L' = 0: ((1-|1-0|/(1+1))*0.1*1.25 = **0.0625** puncte
- S = 4, S' = 3: ((1-|4-3|/(4+1))*0.3*1.25 = **0.3** puncte
- R = 5, R' = 3: ((1-|5-3|/(5+1))*0.3*1.25 = **0.25** puncte.

Așadar, pentru prima echipă se vor acorda 0.125 + 0 + 0.0625 + 0.3 + 0.25 = **0.7375** puncte.

Pentru a doua echipă avem:

- W = 1, W' = 0: ((1-|1-0|/(1+1))*0.2*1.25 = **0.125** puncte
- D = 2, D' = 3: ((1-|2-3|/(2+1))*0.1*1.25 = **0.08333** puncte
- L = 0, L' = 0: 0.1*1.25 = **0.125** puncte
- S = 5, S' = 3: ((1-|5-3|/(5+1))*0.3*1.25 = **0.25** puncte
- R = 3, R' = 3: 0.3*1.25 = **0.375** puncte.

Așadar, pentru a doua echipă se vor acorda 0.125 + 0.08333 + 0.125 + 0.25 + 0.375 = **0.95833** puncte.

Pentru a treia echipă avem:

- W = 2, W' = 0: ((1-|2-0|/(2+1))*0.2*1.25 = **0.08333** puncte
- D = 1, D' = 3: ((1-|1-3|/(1+1))*0.1*1.25 = **0** puncte
- L = 0, L' = 0: 0.1*1.25 = **0.125** puncte
- S = 7, S' = 3: ((1-|7-3|/(7+1))*0.3*1.25 = **0.1875** puncte
- R = 4, R' = 3: ((1-|4-3|/(4+1))*0.3*1.25 = **0.3** puncte.

Așadar, pentru a treia echipă se vor acorda 0.08333 + 0 + 0.125 + 0.1875 + 0.3 = **0.69583** puncte.

Pentru a patra echipă avem:

- W = 0, W' = 0: 0.2*1.25 = **0.25** puncte.
- D = 0, D' = 3: ((1-|0-3|/(0+1))*0.1*1.25 < **0** puncte
- L = 3, L' = 0: ((1-|3-0|/(3+1))*0.1*1.25 = **0.03125** puncte
- S = 2, S' = 3: ((1-|2-3|/(2+1))*0.3*1.25 = **0.25** puncte
- R = 6, R' = 3: ((1-|6-3|/(6+1))*0.3*1.25 = **0.21429** puncte.

Așadar, pentru a patra echipă se vor acorda 0.25 + 0 + 0.03125 + 0.25 + 0.21429 = **0.74554** puncte.

În concluzie, pentru acest test se vor acorda 0.7375 + 0.95833 + 0.69583 + 0.74554 = **3.13** puncte (valoare obținută după trunchiere).

Timp de execuție: 1 secundă/test

P040317: Atlantis

Exploratorul *Iahim Uratrox* face o nouă vizită în capitala mitică a *Atlantidei*. Aici descoperă o piramidă ciudată a cărei intrare este, din nefericire, protejată printr-un nou tip de mecanism.

Datorită experienței acumulate, el observă deja că în fața piramidei se află mai multe comutatoare. Lângă fiecare comutator se află câte o plăcuță pe care este înscris un semn reprezentând un număr natural strict pozitiv. Pe o plăcuță mai mare este înscris un alt număr.

Iahim Uratrox a furnizat aceste informații cercetătorilor care au început să studieze secretul acestui nou tip de mecanism creat de atlanți cu mii de ani în urmă.

Cercetătorii au descoperit destul de repede (sunt și ei mai experimentați) că, pentru a debloca intrarea, trebuie activate mai multe comutatoare (cel puțin două), astfel încât suma numerelor de pe plăcuțele corespunzătoare acestora să fie egală cu valoarea înscrisă pe plăcuța mai mare.

Cu toate acestea, s-a dovedit că nu oricare astfel de alegere duce la deschiderea intrării în piramidă. Nu s-au descoperit alte detalii referitoare la mecanism, motiv pentru care *Iahim Uratrox* dorește să știe câte posibilități de a alege comutatoarele există pentru a ști câte șanse are să determine configurația, despre care se crede că este unică, ce va deschide intrarea.

Date de intrare

Fișierele de intrare sunt deschise și pot fi descărcate folosind *link*-urile de pe pagina dedicată problemei; adresa acesteia este **http://www.ginfo.ro/concurs/runda19/index.shtml**. Numele fișierelor au forma **ATLANTISXX.IN**, unde **XX** ia valori între **01** și **20** și reprezintă numărul testului.

Fiecare dintre cele 20 de fișiere de intrare conține pe prima linie numărul *N* al plăcuțelor "mici" și valoarea *S* inscripționată pe plăcuța mai mare.

Următoarele *N* linii vor conține numerele inscripționate pe cele *N* plăcuțe.

Date de ieșire

Pentru această problemă nu va trebui să scrieți un program, ci să generați doar cele 20 de fișiere de ieșire corecte. Acestea vor fi denumite **ATLANTISXX.OUT**, unde **XX** ia valori între 01 și 20 și reprezintă numărul testului.

Prima linie a unuia dintre fișierele de ieșire va conține textul ATLANTIS - TEST #XX, unde XX reprezintă numărul testului.

Cea de-a doua linie va conține un singur număr care va indica numărul posibilităților de a alege cel puțin două comutatoare astfel încât suma numerelor de pe plăcuțele corespunzătoare acestor comutatoare să fie egală cu valoarea de pe plăcuța mai mare.

Observație

Datorită faptului că denumirile fișierelor de ieșire trebuie să conțină 12 caractere, aceste fișiere nu vor putea fi create folosind sistemul de operare *MS-DOS*.

Exemplu

Vom considera că acest exemplu reprezintă testul **00**.

ATLANTIS00.IN

```
5 5
1
2
3
4
5
```

ATLANTIS00.OUT

```
2
```

P040318: Maxime

Se consideră un șir *a* cu *N* elemente ale căror valori sunt numere întregi distincte care nu sunt cunoscute.

Va trebui să determinați indicii celor mai mari trei numere din acest șir. Pentru determinarea acestora, puteți efectua un număr limitat de comparări între două elemente ale șirului *a*.

Pentru aceasta, aveți la dispoziție o bibliotecă externă, care vă pune la dispoziție rutine pentru:

- inițializare;
- determinarea numărului de elemente ale șirului;
- determinarea numărului maxim de comparări pe care le aveți la dispoziție;
- compararea a două elemente ale șirului *a*;
- furnizarea ca rezultat a celor trei numere;
- încheierea execuției programului.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca, va trebui să includeți în programul dumneavoastră istrucțiunea **uses** `max`; pentru limbajul *Pascal* și `#include "max.h"` pentru limbajul *C/C++*.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

procedure `Init;`
void `Init()`

- trebuie apelată la începutul programului și este folosită de către bibliotecă pentru inițializare;
- întrerupe execuția programului dacă este apelată a doua oară.

function `GetN:Integer;`
int `GetN()`

- returnează numărul de elemente (*N*) ale șirului *a*.

function `GetNoComp:Integer;`
int `GetNoComp()`

- returnează numărul de comparări (apeluri ale funcției `IsGreater`) pe care le mai aveți la dispoziție (în momentul apelului funcției `GetNoComp`) pentru determinarea permutării *p*.

function `IsGreater(i,j:Integer):Boolean;`
int `IsGreater(int i,int j)`

- returnează `true` (valoare nenulă) dacă *a*[*i*] > a[*j*] și `false` (0) în caz contrar;
- întrerupe execuția programului dacă cel puțin una dintre valorile `i` și `j` nu este un număr întreg cuprins între 1 și *N* sau dacă numărul de apeluri ale acestei funcții depășește numărul maxim permis.

procedure `Max1(x:Integer);`
void `Max1(int x)`

- acceptă ca rezultat faptul că cel mai mare element al șirului se află pe poziția x;
- întrerupe execuția programului dacă este apelată a doua oară sau dacă valoarea x nu este cea corectă.

procedure `Max2(x:Integer);`
void `Max2(int x)`

- acceptă ca rezultat faptul că cel mai mare element al șirului obținut după eliminarea maximului se află pe poziția x;
- întrerupe execuția programului dacă este apelată a doua oară sau dacă valoarea x nu este cea corectă.

procedure `Max3(x:Integer);`
void `Max3(int x)`

- acceptă ca rezultat faptul că cel mai mare element al șirului obținut după eliminarea celor mai mari două numere se află pe poziția x;
- întrerupe execuția programului dacă este apelată a doua oară sau dacă valoarea x nu este cea corectă.

procedure `EndProgram;`
void `EndProgram(void)`

- întrerupe execuția programului.

Toate funcțiile și procedurile (excepție: `Init`) întrerup execuția programului dacă sunt apelate înaintea inițializării.

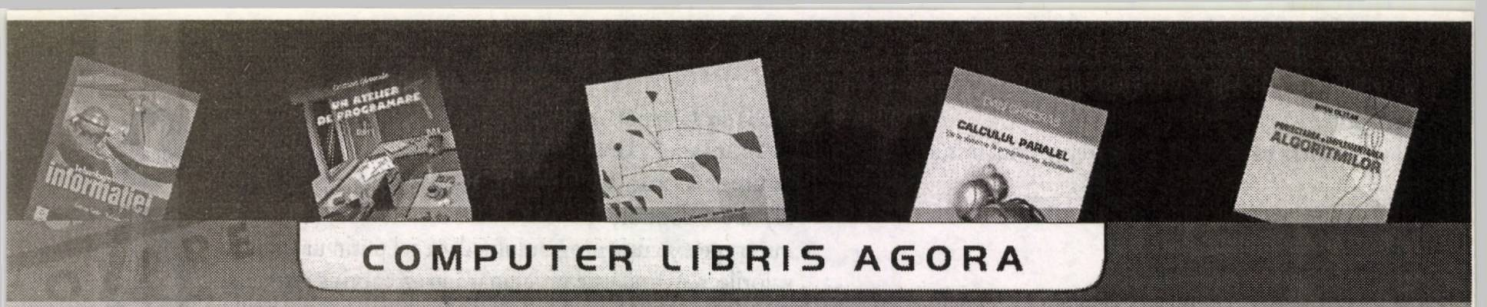
Restricții și precizări

- Programul vostru nu va citi date din nici un fișier de intrare și nu va scrie date în nici un fișier de ieșire.
- $3 \le N \le 255$.

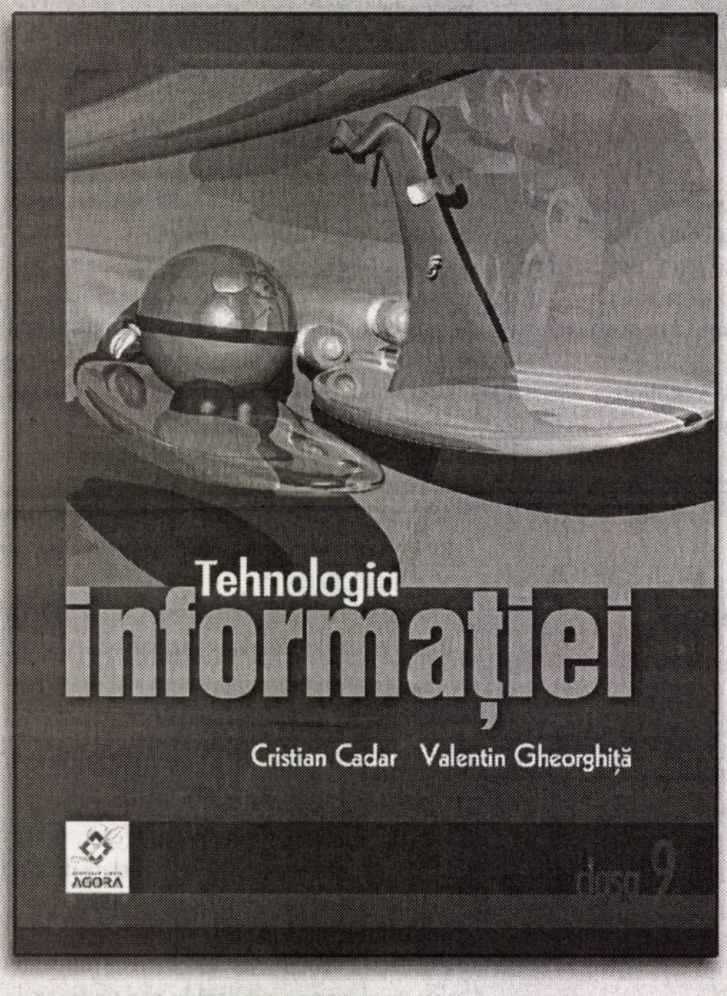
Exemple de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
`Init`	-
`GetN`	3
`GetNoComp`	3
`IsGreater(1,2)`	`true (1)`
`IsGreater(1,3)`	`false (0)`
`IsGreater(2,3)`	`false (0)`
`Max1(1)`	-
`Max2(3)`	-
`Max3(2)`	-
`EndProgram`	-

Timp de execuție: 1 secundă/test



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel.0264 - 192422

Despre CONCURSURI

Mihai Stroe

Articolul de față își propune să vină în ajutorul elevilor care doresc să obțină performanțe la concursurile de informatică. Observațiile din cadrul acestui articol sunt, în mare parte, rezultatul experienței autorului și a altor participanți la concursurile de informatică; ele nu constituie "axiome", ci mai curând sfaturi, valabile atât pentru elevii aflați la început, cât și pentru cei cu o mai mare experiență și participări la concursuri importante.

Concursurile de programare

O primă întrebare la care doresc să afle răspunsul cei care vor să participe la olimpiadele de informatică este: "*Care sunt cele mai importante concursuri de programare?*". În cele ce urmează vom încerca să răspundem la această întrebare.

IOI

Bineînțeles, pentru a răspunde la întrebare trebuie să începem cu *Olimpiada Internațională de Informatică*. Acest concurs reunește anual elevi de liceu din diferite țări ale lumii. Fiecare țară este reprezentată de cel mult patru concurenți, iar numărul țărilor participante crește în fiecare an. Pentru ediția 2003 se preconizează că vor participa aproximativ 90 de țări.

Concursul este individual și se desfășoară sub forma a două probe. La fiecare probă concurenții au la dispoziție cinci ore pentru a rezolva trei probleme cu un grad ridicat de dificultate. După fiecare probă, programele concurenților sunt evaluate automat, cu ajutorul unor programe de evaluare. După cele două zile de concurs se stabilește clasamentul final și se acordă medaliile. Rezultatele sunt secrete până în momentul decernării premiilor. Se stabilesc și clasamente neoficiale pe națiuni (în funcție de premii și în funcție de medalii).

La aproape toate edițiile acestui concurs, echipa *României* a avut rezultate excelente. Astfel, începând din anul 1993, cu doar două excepții, toți cei patru componenți ai echipei țării noastre au obținut medalii; conform punctajelor individuale, s-au obținut două locuri I (1993 și 1998) cu punctaj maxim și un loc II (2001), iar în clasamentul pe națiuni *România* este o prezență constantă în primele cinci locuri.

Din nefericire, se observă o ușoară tendință descendentă a rezultatelor echipei noastre; la ultima ediție, după mult timp, unul dintre concurenții români nu a obținut nici o medalie.

Mulți se întreabă cum a fost posibilă obținerea acestor rezultate în condițiile în care știm cu toții că situația învățământului românesc este destul de precară. Totul s-a realizat prin munca susținută a elevilor, profesorilor pregătitori, membrilor *Comisiei Naționale* etc. De la an la an, elevilor li se oferă mai multe șanse de afirmare, apar noi concursuri, iar cele vechi sunt mai bine puse la punct, se încearcă adaptarea concursurilor la standardele internaționale etc.

BOI și CEOI

O dovadă a acestei preocupări este inițierea de către țara noastră a două concursuri internaționale regionale. Primul dintre ele este *Olimpiada de Informatică a Europei Centrale* (*CEOI*) a cărei primă ediție a avut loc la *Cluj-Napoca* în anul 1994. Ediția din anul 2000 a acestui concurs a avut loc tot la *Cluj-Napoca*. Standardele acestui concurs sunt mai ridicate decât cele de la *IOI*, deoarece elevii din țările participante sunt întotdeauna foarte bine pregătiți. *CEOI* 2003 se va desfășura la *Munster*, în *Germania*.

Un alt concurs important este *Balcaniada de Informatică* (*BOI*) a cărei primă ediție a avut loc la *Constanța* în anul 1993. Până acum câțiva ani, standardele acestui concurs nu sunt foarte ridicate, iar concurenții români se clasau întotdeauna pe primele locuri. Cel mai concludent exemplu în acest sens este faptul că la ediția din anul 2001 toate cele patru medalii de aur au fost obținute de reprezentanți ai *României*. În ultimii ani se înregistrează o creștere a nivelului de dificultate a acestui concurs. Ediția din 2003, care se așteaptă să aibă un nivel ridicat de dificultate, va fi organizată de țara noastră, la *Iași*.

Mai există și alte concursuri internaționale, cum ar fi *Olimpiada Țărilor Baltice*, la care *România* nu participă.

Problemele propuse spre rezolvare la aceste concursuri pot fi folosite cu succes în cadrul pregătirii.

Concursuri naționale

Pentru a participa la un concurs internațional, un elev trebuie să parcurgă mai multe etape. O parte dintre acestea (cele obligatorii) sunt: participarea la olimpiadele locale și județene, calificarea la *Olimpiada Națională*, obținerea unui rezultat care să permită participarea la barajele pentru selecția în lotul național, obținerea unuia dintre primele 15 locuri la aceste baraje, apoi obținerea unor rezultate bune în cadrul lotului.

Lotul național lărgit (în componența căruia se află aproximativ 15 elevi) participă la mai multe pregătiri, în cadrul cărora sunt incluse câteva baraje pentru selectarea primilor patru elevi (care vor participa la *Olimpiada Internațională*) și formarea echipelor pentru *CEOI* și *BOI*.

Alte concursuri

Probabil, cea mai importantă etapă care trebuie parcursă este pregătirea, care ar trebui să se desfășoare tot timpul anului, nu numai în perioada premergătoare concursurilor.

Există câteva concursuri regionale și naționale similare olimpiadelor (diferă timpul de concurs, numărul de probe și dificultatea problemelor). Câteva dintre acestea sunt:

- *Marele Premiu al Palatului Copiilor* - concurs organizat la *Palatul Național al Copiilor* din *București*; participă echipe ale *Cluburilor Copiilor* din mai multe județe ale țării;
- *Concursul "Grigore C. Moisil"* organizat anual la *Lugoj* - are o desfășurare similară;
- diferite concursuri interjudețene organizate în anumite regiuni ale țării; mai multe astfel de concursuri poartă numele lui *Grigore Moisil*; alte concursuri sunt *Info' Oltenia*, *LInfo@SV*, *Urmașii lui Moisil* etc.

De obicei, aceste manifestări sunt mai ample; la aceste concursuri există și alte secțiuni, cum ar fi cele dedicate dezvoltării de aplicații *software*. Există și alte concursuri destinate exclusiv aplicațiilor *software* (de exemplu concursul *Microinformatica* de la *Cluj-Napoca*), care sunt și ele utile în pregătirea pentru olimpiade, deoarece crearea aplicațiilor ajută la dezvoltarea gândirii algoritmice.

Pe plan național au apărut în ultimii ani câteva concursuri la care problemele se rezolvă acasă. În cazul acestora, problemele sunt publicate în anumite reviste și / sau pe *site*-uri *Web*; rezolvările se trimit (de obicei prin *e-mail*) înaintea expirării unui anumit termen. Există câteva runde de acest tip, după care primii clasați se întâlnesc la "marea finală", care se desfășoară sub forma unei probe de concurs care este similară cu cea de la olimpiade.

Unul dintre cele mai cunoscute concursuri de acest tip este cel organizat de revista noastră.

Un alt concurs similar este *Cupa Fujitsu-Siemens* (fosta *Cupa Compaq*, a intervenit o schimbare a sponsorilor), concurs care este organizat începând cu anul 1995. Până la ediția 2002, acest concurs consta în două runde desfășurate "la distanță" și o etapă finală. Enunțurile problemelor de la primele două runde erau publicate în revista *PC World*. La ultima ediție, finaliștii *Cupei Fujitsu-Siemens* au fost primii concurenți de la *OlimpiadaOnline* (concurs cu o organizare similară concursului de programare *Bursele Agora* de anul trecut). Din nefericire, se pare că *site*-ul **www.olimpiadaonline.ro** nu mai este accesibil.

În acest an a fost inițiat un nou *site* de pregătire și concursuri, care poate fi accesat la **www.fudv.ro/campion**. Problemele de pe acest *site* sunt propuse de membri ai *Comisiei Olimpiadei Naționale*. Considerăm acest *site* o resursă valoroasă pentru pregătire.

Recomandăm participarea la acest tip de concursuri, atât pentru valoarea premiilor puse în joc, cât și ca modalitate de antrenament.

Ultima categorie de concursuri la care ne oprim sunt concursurile *online* organizate în afara *României*, la care puteți participa prin intermediul *Internet*-ului.

În acest caz, problemele sunt disponibile un anumit interval de timp (câteva ore, cel mult zile), în care se pot trimite rezolvări. Aceste concursuri constituie o resursă importantă de probleme pentru pregătire.

Cele mai importante *site*-uri care oferă concursuri *online* sunt **acm.uva.es** și **acm.timus.ru**, care tratează concursurile studențești *ACM*. Ambele *site*-uri vă pun la dispoziție și câte o arhivă foarte bogată de probleme și pot fi folosite cu succes în pregătirea pentru olimpiade.

Un alt concurs *online* important este *USACO* (*USA Computing Olympiad*), prin intermediul căruia se selectează lotul național lărgit al *SUA*. Concursul constă în mai multe faze, desfășurate pe parcursul anului. Prin bunăvoința organizatorilor, concursul a devenit internațional, iar ultimele ediții au adus ca noutate traducerea problemelor în mai multe limbi, printre care, uneori, și limba română. Deoarece *SUA* va organiza ediția din acest an a *Olimpiadei Internaționale*, în ultimul timp tot mai mulți elevi români au participat la concursurile *USACO*, obținând rezultate foarte bune.

Merită menționat și *Internet Problem Solving Contest*, concurs organizat o dată pe an. În cadrul acestui concurs se pun la dispoziție enunțurile problemelor și fișierele de intrare și se așteaptă fișiere de ieșire, fără a se cere și programele care rezolvă problemele.

Nu putem acoperi integral toate concursurile disponibile pe *Internet* în acest articol. Există multe alte concursuri; este foarte importantă citirea regulamentului înainte de rezolvarea problemelor, deoarece pot exista restricții care diferă de la caz la caz.

De multe ori, încălcarea unor anumite prevederi ale regulamentului conduce la descalificare și la interzicerea participării la edițiile viitoare.

Datorită faptului că majoritatea acestor concursuri sunt internaționale, cunoașterea limbii engleze devine o necesitate pentru a putea participa.

Pregătirea pentru concursuri

Prima observație este aceea că pregătirea trebuie să fie un proces cu o anumită continuitate. O pregătire intensă înaintea unui concurs este foarte importantă deoarece îmbunătățește viteza de implementare și reduce riscul apariției erorilor, dar nu poate rezolva anumite lacune teoretice.

Există mai multe aspecte ale pregătirii:

- organizarea globală a pregătirii;
- pregătirea teoretică;
- simularea unor probe de concurs;
- discuțiile cu alți elevi și profesori referitor la anumite probleme;
- pregătirea psihică;
- pregătirea de la locul desfășurării probei.

Organizarea globală a pregătirii

Acest aspect nu făcea parte din lista publicată anul trecut, însă poate contribui la obținerea unor rezultate excelente.

O parte foarte importantă a unei pregătiri sistematice constă în elaborarea unei liste cu metodele și tehnicile cunoscute și necunoscute, punctele slabe etc.

Lista trebuie să conțină algoritmii care apar în mod frecvent în cadrul problemelor de concurs, problemele care apar la implementare, la organizarea timpului în concurs etc.

Conținutul listei se modifică în timp; o parte din algoritmii necunoscuți sunt învățați și devin cunoscuți, se descoperă noi puncte slabe, se evidențiază existența unor algoritmi necunoscuți care trebuie învățați etc. Aceasta este o modalitate excelentă de a măsura progresul și de a găsi noi direcții de urmat în pregătire.

Vom încerca, într-unul din articolele viitoare, să oferim un astfel de exemplu de abordare sistematică.

Pregătirea teoretică

Majoritatea problemelor propuse spre rezolvare la concursuri depășesc nivelul manualelor de informatică. De exemplu, deși se propun o mulțime de probleme a căror rezolvare implică deținerea de cunoștințe din domeniul teoriei grafurilor, nu toți algoritmii necesari sunt cuprinși în programa de învățământ.

Pentru a rezolva lacunele teoretice, este necesară studierea unor cărți care să acopere un spațiu teoretic cât mai vast. "*Biblia*" algoritmilor este considerată așa-numita *CLR*. Denumirea provine de la inițialele numelor celor trei autori (*Cormen*, *Leiserson*, *Rivest*); titlul cărții este *Introducere în algoritmi*.

Până acum câțiva ani cartea era foarte greu de găsit și puținii norocoși care reușeau să o obțină aveau un avantaj important la aceste concursuri. Din fericire, între timp a apărut traducerea în limba română a acestei cărți și acum oricine este interesat o poate achiziționa.

Câteva dintre cărțile care ar trebui parcurse sunt cele scrise de foști olimpici; prin intermediul acestora, autorii vă împărtășesc o parte din experiența acumulată. Cele mai cunoscute astfel de cărți sunt:

- *Proiectarea și implementarea algoritmilor* - scrisă de *Mihai Oltean* și apărută la editura *Computer Libris Agora din Cluj-Napoca*;
- *Culegere de probleme și programe PASCAL* - scrisă de autorul acestui articol în colaborare cu *Cristian Cadar* și apărută la editura *Petrion* din *București*;
- *Psihologia concursurilor de informatică* - scrisă de *Cătălin Frâncu* și apărută la editura *L&S* din *București*.

În afară de cărțile menționate, *Internet*-ul se dovedește din nou o resursă foarte importantă. O căutare pe *Internet* poate localiza informații interesante: descrierea unor anumiți algoritmi împreună cu performanțele lor, tratări ale unor probleme clasice prin mai multe metode etc.

Simularea unor probe de concurs

Cea mai potrivită modalitate de pregătire pentru a face față unei situații este simularea ei, adică tratarea unei situații asemănătoare. De exemplu, un fotbalist care execută foarte bine loviturile libere la antrenamente are șanse mari să le execute la fel de bine și în timpul meciului, deoarece este bine pregătit pentru această situație. Același principiu se aplică și în domeniul concursurilor de programare. Dacă vă pregătiți pentru un concurs, citirea unor cărți nu este suficientă! Trebuie să vă analizați comportamentul în situații similare.

Situația cea mai asemănătoare unui anumit concurs este un alt concurs de același tip! Participarea la N concursuri crește șansele obținerii unui rezultat mai bun la al N+1-lea.

Concursurile de pe *Internet* sunt destul de dese și sunt organizate foarte bine. Din nefericire, nivelul de dificultate al acestor concursuri nu este întotdeauna cel dorit de cel care se pregătește.

O posibilitate de simulare a unui concurs este rezolvarea problemelor de la o ediție precedentă! De exemplu, cineva care se pregătește pentru *Olimpiada Națională* din 2003 ar trebui să rezolve problemele date la *Olimpiada Națională* din 2002 la clasa respectivă (există și cazuri în care simulări de acest tip nu sunt foarte concludente; de exemplu, între 2000 și 2001 programa școlară a suferit câteva variații, care s-au reflectat în tipul problemelor de la clasa a X-a).

Pentru ca simularea să reflecte cât mai bine realitatea, problemele se rezolvă în timpul stabilit, fără pauze, la prima citire (sau, dacă au fost citite anterior, nu se recitesc în zilele premergătoare simulării).

Experiența obținută este apropiată de cea a concursului propriu-zis, dar lipsește stres-ul care apare, inevitabil, în timpul competiției.

Este important ca, după fiecare simulare sau concurs real, să vă analizați comportarea și să învățați din eventualele greșeli de abordare (de exemplu, puteți ajunge la concluzii de genul "*Această problemă trebuia abordată prima*" sau "*Nu am citit integral enunțul și am rezolvat o altă problemă*").

Această analiză se încadrează și în cadrul primei direcții, "*Organizarea globală a pregătirii*"; concluziile analizei duc la detectarea acțiunilor necesare pentru a îmbunătăți situația.

În plus, se recomandă notarea celor mai frecvente greșeli de implementare și examinarea periodică a listei (inclusiv în timpul depanării programelor, în cadrul simulării concursurilor); în acest fel, greșelile respective vor dispărea în timp.

Discuțiile cu alți elevi și profesori

Puteți învăța foarte mult de la profesori sau elevi mai experimentați! Există exemple de elevi pentru care a contat foarte mult pregătirea individuală, dar majoritatea au fost ajutați de pregătirea organizată, în grupuri de elevi, sub îndrumarea unor profesori cu preocupări de acest gen.

În cadrul pregătirilor de acest tip se discută algoritmi, se propun probleme spre rezolvare, se discută diversele modalități de rezolvare, se obțin mai multe informații despre concursuri etc. În plus, elevii aduc în discuție diverse probleme cu care s-au întâlnit în cadrul pregătirii individuale.

Dacă doriți să participați la astfel de pregătiri, trebuie să luați legătura cu alți elevi interesați de concursuri, sau cu profesorii care se ocupă de pregătirea elevilor pentru olimpiade. În prezent se organizează pregătiri la nivel de liceu, oraș, județ etc. Astfel de pregătiri cresc valoarea tuturor participanților, deci sunt foarte importante.

Pregătirile organizate au luat amploare odată cu înființarea centrelor de excelență.

Pregătirea psihică

Nu vom face aici un tratat de psihologie. În schimb vom formula câteva observații de bun-simț și vom desființa anumite prejudecăți.

Este cunoscut faptul că o atitudine mentală pozitivă este cheia succesului în cele mai multe situații. Din nefericire, unii concurenți încep proba cu un moral nu tocmai ridicat.

Iată câteva din falsele probleme cu care se confruntă anumiți concurenți:

- *Participă și X, care e mai bun ca mine, deci nu am nici o șansă să câștig!*

  Fals! Nu s-a demonstrat că *X* este mai bun, cel mult a obținut rezultate mai bune până acum și poate avea șanse mai mari. Problemele din concursul curent sunt aceleași pentru toți, condițiile de desfășurare sunt aceleași și antecedentele nu contează. Totul se reia de la zero.

  În plus, participarea la un concurs puternic poate aduce mai multă experiență pentru viitor. Pentru a ajunge la valoarea necesară câștigării unor concursuri, trebuie să participați la cât mai multe și să le tratați cu seriozitate.
- *Am obținut prea puține puncte în prima zi, nu mai am nici o șansă la premii!*

  Problemele de la concursurile cu mai multe probe sunt, în general, destul de dificile. De multe ori, la *Olimpiada Națională* sau la concursurile internaționale, obținerea a jumătate din punctele puse în joc înseamnă câștigarea unui premiu.

  Dacă în prima zi rezultatele obținute sunt nesatisfăcătoare, un rezultat foarte bun în ziua a doua poate aduce premiul dorit. Reamintim că, anul trecut, pentru intrarea în lotul național lărgit, la barajele de selecție, a fost suficientă obținerea a mai puțin de 200 puncte, dintre cele 600 posibile!
- *Comisia nu este imparțială!*

  Deși foarte rar, unii elevi susțin acest lucru. Mulți alții nu au curajul să afirme așa ceva, dar sunt convinși că nu au aceleași șanse ca și preferații comisiei. Puteți fi siguri că în cazul concursurilor de informatică (cele care se respectă) comisia este imparțială și subiectivismul este eliminat datorită evaluării automate. Dacă punctajul nu pare a fi în concordanță cu cel așteptat, depuneți contestații și cereți lămuriri! Dacă aveți dreptate, veți primi punctajul corect.
- *Nu sunt suficient de bine pregătit!*

  Această apreciere este, uneori, mai realistă. Totuși, trebuie să știți că o foarte mare importanță o are inspirația de moment sau norocul (poate problemele vor fi similare cu unele rezolvate anterior).

Evident, optimismul exagerat poate, la rândul său, să fie dăunător. Cel mai bine ar fi să adoptați atitudinea cea mai potrivită pentru propria personalitate. Veți observa, în timp, care este aceasta.

Pregătirea psihică nu are legătură numai cu concursul propriu-zis. O atitudine mentală pozitivă, de învingător, este utilă pe tot timpul pregătirii pentru concursuri.

Pregătirea de la locul desfășurării probei

Deși ar putea părea bizar, acest aspect este foarte important, dar este neglijat de multe ori de către concurenți. Această pregătire constă în:

- somn odihnitor în noaptea care precede ziua concursului;
- obținerea atitudinii mentale dorite;
- prezentarea la timp în sală, cu toate obiectele necesare.

Există o mulțime de obiecte pe care ar trebui să le aveți la voi în timpul desfășurării probei. Vom încerca să le amintim pe cele mai importante.

În primul rând trebuie să aveți un ceas pentru a ști întotdeauna cât timp mai aveți la dispoziție. Nu este bine să vă bazați doar pe ceasul calculatorului, deoarece consultarea acestuia necesită timp care nu mai poate fi folosit pentru rezolvarea problemelor.

Trebuie să aveți instrumente de scris, foi albe și foi cu pătrățele (pentru a rezolva problemele care necesită cunoștințe de geometrie analitică). De obicei, organizatorii vă pun la dispoziție foi albe, dar este bine să nu riscați.

Nu în ultimul rând, ar fi bine să aveți o sticlă de suc și o ciocolată sau un croissant; din nefericire, concursul înce-

pe deseori cu întârziere și este bine ca foamea sau setea să nu vă preocupe în timpul rezolvării problemelor.

Concursul propriu-zis

De acum vom presupune că vă aflați în fața calculatorului și primiți problemele. Ne vom concentra, în principal, asupra concursurilor de tipul olimpiadelor naționale și internaționale; totuși, majoritatea recomandărilor sunt valabile și pentru alte tipuri de concursuri.

De obicei, veți primi problemele listate pe foi. Pentru început va trebui să verificați dacă ați primit toate problemele. În situația, puțin probabilă, în care lipsesc anumite foi, trebuie să anunțați imediat acest lucru.

După primirea subiectelor, veți începe lecturarea lor. La începutul probei trebuie să citiți integral toate problemele pentru a vă forma o imagine de ansamblu asupra acestora și pentru a descoperi eventuale ambiguități. Problemele se citesc integral, toate la începutul probei. Nu este recomandat să începeți implementarea imediat ce aveți o idee de rezolvare pentru una dintre probleme.

Este foarte important să citiți toate enunțurile. Altfel spus, în primele zece minute (sau mai mult) nu se atinge calculatorul. Întotdeauna, când citiți o problemă, este indicat să întoarceți foaia pentru a vedea dacă enunțul continuă și pe verso.

De obicei, în primele 30 sau 60 de minute ale concursului pot fi adresate întrebări comisiei, pentru a clarifica eventualele ambiguități din enunțuri. Acestea sunt redactate în scris, foile sunt preluate de supraveghetorul din sală și trimise la comisie.

Răspunsul s-ar putea să întârzie, deci este indicat să nu irosiți timpul așteptând răspunsul fără a mai face nimic altceva. Puteți fie să vă gândiți la rezolvarea unei probleme, fie să începeți să implementați (dacă există ceva ușor de implementat, cum ar fi o problemă simplă sau o rutină pentru citirea datelor de intrare).

În majoritatea situațiilor, întrebările trebuie formulate în așa fel încât răspunsul să fie "*Da*" sau "*Nu*". Dacă întrebarea nu este astfel exprimată sau dacă răspunsul se găsește în textul problemei, veți primi răspunsul "*Fără comentarii*", caz în care va trebui mai întâi să studiați corectitudinea întrebării și, dacă aceasta este corect formulată, să recitiți enunțul problemei.

Concurenții trebuie să profite cât mai mult de această perioadă, pentru a clarifica eventualele nelămuriri. Pentru a rezolva problemele trebuie, în primul rând, să știți care sunt cerințele.

Este foarte important ca la întrebări să răspundă membrii comisiei. Au existat unele cazuri în care supraveghetorii își dădeau cu părerea asupra enunțului; dacă aceștia nu fac parte din comisie, răspunsul s-ar putea să nu fie cel corect. Nu trebuie să acceptați răspunsuri dacă acestea nu sunt însoțite de semnătura unui membru al comisiei.

După ce toate problemele sunt clare, se trece la căutarea algoritmilor de rezolvare. Și în această etapă, se vor aborda toate problemele, dar analiza va fi rapidă. Problemele pentru care nu se găsește algoritmul de rezolvare vor fi abordate din nou mai târziu.

Enunțul problemei oferă și indicații asupra ordinului de complexitate al algoritmului de rezolvare prin limitele asociate datelor de intrare și precizarea timpului de execuție. Dacă aceste informații nu sunt specificate, va trebui neapărat să semnalați acest lucru prin întrebări adresate comisiei.

Dimensiunea datelor de intrare oferă indicații asupra complexității rezolvării așteptate. De exemplu, dacă datele de intrare constau într-un graf cu 1000 de noduri și 5000 de muchii, se așteaptă un algoritm cu ordinul de complexitate cel mult pătratic. Este probabil ca algoritmul căutat să fie liniar sau liniar-logaritmic.

După această fază, aveți deja o primă idee asupra:

- problemelor și algoritmilor de rezolvare;
- punctajului așteptat (de exemplu, un *backtracking* pentru $N = 1000$ vă va aduce, în general, foarte puține puncte);
- timpului necesar pentru implementarea algoritmilor de rezolvare pentru anumite probleme.

Dacă există o problemă simplă, care se poate rezolva foarte repede, este bine să începeți cu aceasta! O problemă rezolvată foarte repede vă oferă destul de mult timp pentru restul concursului (de multe ori rezolvarea corectă a două probleme din trei înseamnă obținerea unui rezultat excelent).

După rezolvarea problemelor simple, se analizează restul problemelor. De obicei, în această fază există probleme pentru care concurentul nu are nici o idee de rezolvare (unele dintre acestea pot deveni simple printr-o analiză mai atentă), probleme a căror implementare necesită mult timp și probleme pentru care ideea de rezolvare curentă nu va aduce punctajul maxim.

Indiferent ce problemă decideți să abordați în continuare, țineți cont de următoarele observații:

- scopul concurentului nu este să rezolve corect problemele, ci să obțină cât mai multe puncte; este mult mai bine să implementați în 10-15 minute o rezolvare care poate aduce 10-20% din punctajul maxim, decât să nu rezolvați deloc problema;
- programele finale trebuie să realizeze exact ceea ce intenționați.

După realizarea unui program care rezolvă o anumită problemă, se impune testarea acestuia. Este momentul unei recitiri a enunțului problemei (dacă au existat unele scăpări, acum este timpul să fie înlăturate).

Fiecare program este testat pentru testul simplu din exemplul dat în enunț și pentru cât mai multe teste elaborate manual.

Dacă aveți timp, puteți crea un generator de teste și un evaluator care verifică anumite caracteristici ale soluției obținute de programul vostru.

De exemplu, dacă într-o problemă se cere un drum minim într-un graf, generatorul va genera grafuri aleatoare sau cu anumite caracteristici, iar evaluatorul ar putea verifica dacă muchiile afișate se găsesc în graf, lungimea drumului este cea declarată etc.

Majoritatea concurenților nu vor scrie un generator și un evaluator pentru această problemă; totuși, pentru probleme mai complicate, aceste programe auxiliare sunt de un real ajutor. Astfel, se pot depista unele greșeli de implementare și se estimează timpul de execuție pentru date de intrare mari.

Ratarea unei probleme din cauza unei greșeli de implementare este foarte neplăcută; concurentul pierde puncte importante, pe care le-ar fi putut obține și pe care ceilalți, probabil, le-au obținut.

Un program care nu se încadrează în timp va trebui optimizat. Se recomandă să păstrați o versiune funcțională a acestuia (dacă nu aveți timp să terminați sau greșiți la implementare și nu mai puteți reveni la forma anterioară, ați pierdut foarte mult timp și nu este deloc convenabil să scrieți din nou anumite secvențe din program).

În cazul în care programul depășește cu puțin timpul de execuție, câteva optimizări de cod pot fi suficiente. De exemplu, secvența:

```
for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if a[i,k]+a[k,j]<a[i,j] then
        a[i,j]:=a[i,k]+a[k,j];
```

se poate îmbunătăți observând că `a[i,k]` este citit din memorie de mult mai multe ori decât este necesar. Secvența se rescrie astfel:

```
for k:=1 to n do
  for i:=1 to n do begin
    p:=a[i,k];
    for j:=1 to n do
      if p+a[k,j]<a[i,j] then
        a[i,j]:=p+a[k,j];
  end;
```

Adresarea variabilei `p` este mai rapidă decât a locației `a[i,k]`.

Alte optimizări constau în înlocuirea condițiilor de forma `"if i<=sqrt(n)"` cu `"if i*i<=n"`, a împărțirilor la 2 cu deplasări la dreapta etc. Se pot realiza și optimizări de cod mai importante.

În cazul în care nu știți să rezolvați perfect o problemă, dar ea poate fi abordată prin *backtracking*, cu șanse de obținere a unui punctaj mulțumitor, nu uitați de celelalte alternative: metode euristice *greedy*, algoritmi probabiliști etc. Eventual, puteți combina aceste metode (de exemplu puteți folosi un *greedy* și apoi, până la expirarea timpului, puteți încerca să optimizați prin *backtracking* soluția obținută).

După ce ați terminat problemele (se întâmplă destul de rar să reușiți) nu ieșiți din sală! Este momentul ultimelor teste. La ieșirea din sală trebuie să fiți convinși că ați făcut tot ce era posibil în condițiile date.

Concursul nu s-a terminat încă! Urmează corectarea! Va trebui să verificați punctajul obținut și să fiți pregătit să depuneți o contestație dacă aveți impresia că ceva nu este în regulă.

La unele concursuri, corectarea se face în prezența concurentului; aici aveți ocazia să solicitați să vi se arate testele și ieșirile furnizate de programul vostru, să cereți testarea din afara mediului de evaluare etc.

La alte concursuri, comisia oferă, mai târziu, testele și răspunsurile corecte pentru autoevaluare. Nu ratați ocazia de a vă evalua rezolvările și nu depuneți contestații decât dacă în urma autoevaluării obțineți un punctaj mai mare.

Fiecare concurs este o experiență în plus. Încercați să învățați din greșeli. Discutați, după probă, cu alți concurenți, aflați cum ar fi trebuit rezolvate problemele pe care nu le-ați știut aborda și ce au greșit ceilalți (este bine să învățați și din greșelile altora).

Dacă primele rezultate nu sunt extraordinare, nu trebuie să abandonați! Poate a fost o zi proastă, sau poate ceilalți au mai multă experiență. Situația se va ameliora în timp!

Încheiere

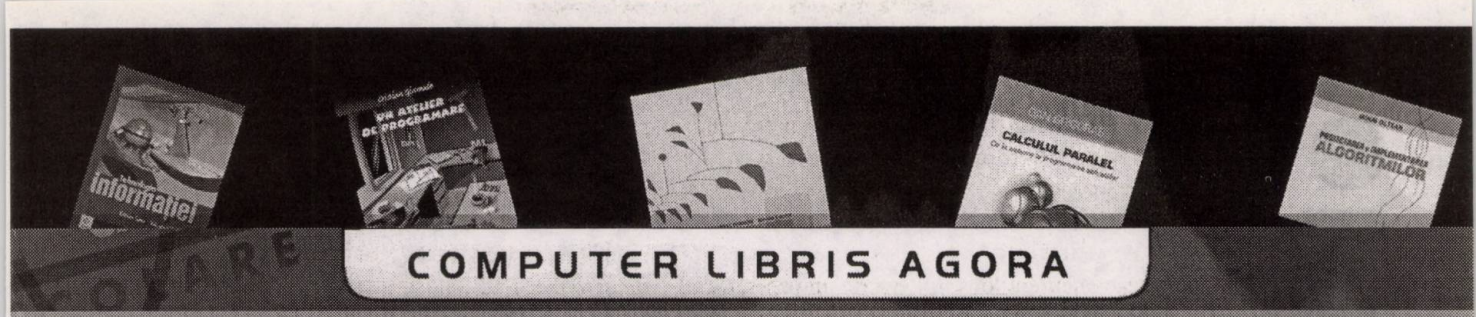
Poate este puțin cam târziu dar, vom încerca să răspundem și la următoarea întrebare: *De ce să particip la astfel de concursuri?*

Principalele avantaje sunt formarea unei gândiri algoritmice, înțelegerea metodelor de rezolvare pentru anumite probleme (care pot apărea și în cadrul dezvoltării de aplicații), dezvoltarea capacității de a reacționa rapid într-un timp scurt. În timp, experiența acumulată în cadrul pregătirii și concursurilor va conduce la eficiență mai mare în cadrul proiectelor de dezvoltare de *software* și, de ce nu, la obținerea unor salarii mai mari la angajare. Deja există companii care iau în considerare participările la concursuri la interviurile pentru angajare.

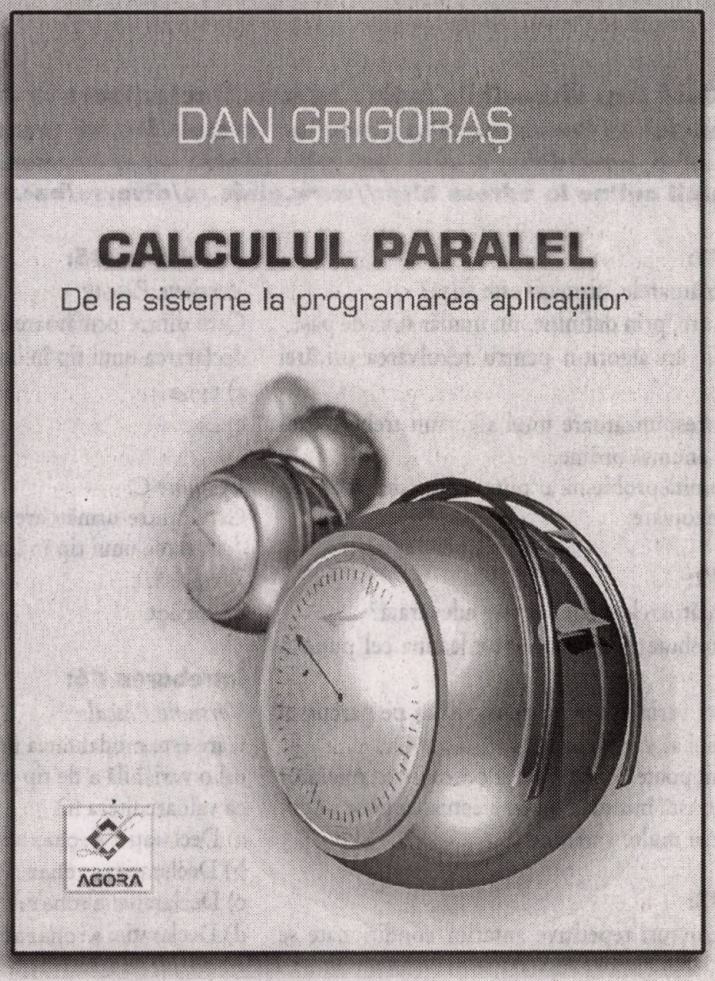
Un avantaj deloc de neglijat este reprezentat de premiile puse în joc, atât cele materiale (calculatoare, diferite componente *hardware*, excursii în străinătate, cărți etc.), cât și intrarea fără examen la facultate, în cazul obținerii unor rezultate bune la olimpiadele naționale sau internaționale.

Încheiem prin a vă ura succes la concursurile la care veți participa. Sperăm că aceste recomandări vă vor fi de un real folos!

Mihai Stroe este student în anul IV la Universitatea Politehnica din București. Poate fi contactat prin e-mail la adresa **mihai_stroe@yahoo.com**.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS
AGORA

BAC INFO' 2003

Chiar dacă sunt disponibile puține informații referitoare la organizarea examenului de Bacalaureat la disciplina informatică, vă prezentăm o variantă orientativă, neoficială, a unui test grilă pentru acest examen. Testul este disponibil online la adresa http://www.ginfo.ro/diverse/bac.shtml.

Întrebarea #1:
Care dintre următoarele afirmații este falsă?
a) Un algoritm are, prin definiție, un număr finit de pași.
b) Poate fi găsit un algoritm pentru rezolvarea oricărei probleme.
c) Operațiile corespunzătoare unui algoritm trebuie executate într-o anumtă ordine.
d) Pentru o anumită problemă ar putea exista mai mulți algoritmi de rezolvare.

Întrebarea #2:
Care dintre următoarele afirmații este adevărată?
a) O expresie trebuie să conțină întotdeauna cel puțin o variabilă
b) Valoarea unei variabile se poate modifica pe parcursul executării unui algoritm
c) O expresie nu poate conține decât cel mult un operator
d) O operație constă întotdeauna în efectuarea unor calcule folosind mai multe variabile

Întrebarea #3:
Corpul unei structuri repetitive anterior condiționate se execută:
a) exact o dată
b) cel mult o dată
c) cel puțin o dată
d) de zero, una sau mai multe ori

Întrebarea #4:
Versiune Pascal:
Care dintre următoarele cuvinte nu este rezervat în limbajul Pascal:
a) `for`
b) `while`
c) `write`
d) `if`

Versiune C:
Care dintre următoarele cuvinte nu este rezervat în limbajul C:
a) `for`
b) `while`
c) `printf`
d) `if`

Întrebarea #5:
Versiune Pascal:
Care dintre următoarele cuvinte cheie este folosit pentru declararea unui tip în limbajul Pascal?
a) `type`
b) `record`
c) `set`
d) `typedef`

Versiune C:
Care dintre următoarele cuvinte cheie este folosit pentru declararea unui tip în limbajul C?
a) `typedef`
b) `record`
c) `struct`
d) `type`

Întrebarea #6:
Versiune Pascal:
Care este modalitatea prin care se declară în limbajul Pascal o variabilă `a` de tip caracter și cum i se atribuie acesteia ca valoare litera `H`?
a) Declarație: `a:character;` Atribuire: `a:='H';`
b) Declarație: `a:char;` Atribuire: `a:='H';`
c) Declarație: `a:char;` Atribuire: `a:=H;`
d) Declarație: `a:character;` Atribuire: `a:=H;.`

Versiune C:
Care este modalitatea prin care se declară în limbajul C o variabilă `a` de tip caracter și cum i se atribuie acesteia ca valoare litera `H`?
a) Declarație: `character a;` Atribuire: `a='H';`
b) Declarație: `char a;` Atribuire: `a='H';`
c) Declarație: `char a;` Atribuire: `a=H;`
d) Declarație: `character a;` Atribuire: `a=H;.`

Întrebarea #7:
Este adevărat că o funcție definită de utilizator trebuie să aibă întotdeauna cel puțin un parametru?
a) Da
b) Nu
c) Da, dar numai dacă nu returnează nici o valoare
d) Da, dar numai dacă returnează cel puțin o valoare

Întrebarea #8:

Versiune Pascal:

Care dintre următoarele instrucțiuni poate fi folosită pentru a atribui unei variabile a codul ASCII al caracterului *?

a) `a := ord('*');` b) `a := '*';`
c) `a := chr('*');` d) `a := ascii(*);`

Versiune C:

Care dintre următoarele instrucțiuni poate fi folosită pentru a atribui unei variabile a codul ASCII al caracterului *?

a) `a = '*';` b) `a = ord('*');`
c) `a = chr('*');` d) `a = ascii(*);`

Întrebarea #9:

Versiune Pascal:

Care dintre următoarele expresii poate fi utilizată pentru a verifica dacă valoarea variabilei a este impară?

a) `a or 1 = 1` b) `a xor 1 = 1`
c) `a and 1 = 1` d) `a not 1 = 1`

Versiune C:

Care dintre următoarele expresii poate fi utilizată pentru a verifica dacă valoarea variabilei a este impară?

a) `a | 1` b) `a ^ 1`
c) `a & 1` d) `a ~ 1`

Întrebarea #10:

Versiune Pascal:

Care dintre următoarele afirmații este falsă în cazul limbajului Pascal?

a) Elementele unui tablou au același tip.
b) Tabloul este o structură statică de date.
c) În declarația unui tablou trebuie folosit cuvântul cheie `array`.
d) Primul element al unui tablou are întotdeauna indicele 1.

Versiune C:

Care dintre următoarele afirmații este falsă în cazul limbajului C?

a) Elementele unui tablou au același tip.
b) Tabloul este o structură statică de date.
c) În declarația unui tablou nu trebuie folosit cuvântul cheie `array`.
d) Primul element al unui tablou are întotdeauna indicele 1.

Întrebarea #11:

Versiune Pascal:

Care dintre următoarele funcții poate fi utilizată pentru determinarea numărului de caractere dintr-un string?

a) `strlen` b) `length` c) `len` d) `size`

Versiune C:

Care dintre următoarele funcții poate fi utilizată pentru determinarea numărului de caractere dintr-un string?

a) `size` b) `length`
c) `strlen` d) `len`

Întrebarea #12:

Versiune Pascal:

Care dintre declarații reprezintă un articol care conține informațiile referitoare la un punct în plan ale cărui coordonate sunt numere reale?

a) `var a:record x,y:real end;`
b) `var a:record begin x,y:real end;`
c) `var x,y:real;`
d) `var a:record a.x,a.y:real end;`

Versiune C:

Care dintre declarații reprezintă un articol care conține informațiile referitoare la un punct în plan ale cărui coordonate sunt numere reale?

a) `struct {float x,y;}a;`
b) `struct {float x} {float y}a;`
c) `float x, y;`
d) `struct {float a.x,a.y;}a;`

Întrebarea #13:

Care dintre următoarele afirmații este adevărată?

a) Un fișier text poate conține doar litere ale alfabetului latin.
b) Înainte de a prelua date dintr-un fișier text, acesta trebuie deschis pentru citire.
c) Nu există posibilitatea de a adăuga litere la sfârșitul unui fișier text.
d) Un fișier text nu poate conține mai mult de 65535 de caractere.

Întrebarea #14:

Versiune Pascal:

Care este cuvântul cheie cu ajutorul căruia se specifică faptul că o anumită variabilă reprezintă un fișier text?

a) `text`
b) `file`
c) `textfile`
d) nu există nici un astfel de cuvânt cheie

Versiune C:

Care este cuvântul cheie cu ajutorul căruia se specifică faptul că o anumită variabilă reprezintă un fișier text?

a) `TEXT`
b) `FILE*`
c) `TEXT_FILE`
d) nu există nici un astfel de cuvânt cheie

Întrebarea #15:

Versiune Pascal:

Prin intermediul cărei proceduri se realizează corespondența dintre o variabilă și un anumit fișier stocat pe disc?

a) `assign` b) `reset`
c) `open` d) `fileopen`

Versiune C:

Prin intermediul cărei funcții se realizează corespondența dintre o variabilă și un anumit fișier stocat pe disc?

a) fopen b) fclose
c) assign d) fileopen

Întrebarea #16:
Ce realizează următorul algoritm descris în pseudocod?

```
citește m
s = 0
cât timp m > 0 execută
  s = s * 10 + rest[m/10]
  m = [m / 10]
sfârșit cât timp
scrie s
```

a) determină suma cifrelor unui număr
b) verifică dacă un număr se divide cu 10
c) determină inversul unui număr (numărul obținut prin citirea cifrelor de la dreapta la stânga)
d) s va avea aceeași valoare cu m

Întrebarea #17:
Care dintre numere va fi recunoscut ca fiind prim (deși, de fapt nu este) dacă se utilizează următorul algoritm de verificare a primalității?

```
citește nr
prim = adevărat
dacă nr < 2 atunci
  prim = fals
sfârșit dacă
dacă rest[nr/2] = 0 atunci
  dacă nr>2 atunci
    prim = fals
  sfârșit dacă
sfârșit dacă
d = 3;
rad = nr^(1/2)          //** rad este radical din nr **//
cât timp (d < rad) și prim execută
  dacă rest[nr/d] = 0 atunci
    prim = fals
  altfel
    d = d + 2
  sfârșit dacă
sfârșit cât timp
dacă prim atunci
  scrie 'Numărul este prim!'
altfel
  scrie 'Numărul nu este prim!'
sfârșit dacă
```

a) algoritmul va intra în ciclu infinit pentru orice număr neprim
b) 17 c) 6 d) 25

Întrebarea #18:
Care este cel mai mic termen al șirului *Fibonacci*, format din cel puțin trei cifre, și care este prim?
a) 233 b) 144 c) 377
d) șirul *Fibonacci* nu conține numere prime.

Întrebarea #19:
Care este elementul neutru al operației de determinare a minimului a două numere?
a) 0 b) ∞ c) -∞
d) această operație nu are element neutru

Întrebarea #20:
În cazul sortării prin inserție a unor numere întregi la fiecare pas se determină:
a) poziția elementului curent în șirul parțial sortat la pașii anteriori
b) elementul minim al șirului nesortat
c) elementul maxim al șirului nesortat
d) nici una dintre cele trei variante anterioare

Întrebarea #21:
Care dintre următoarele condiții trebuie îndeplinite pentru ca două șiruri să poată fi interclasate?
a) șirurile trebuie să aibă același număr de elemente
b) șirurile trebuie să fie ordonate în funcție de un același criteriu
c) numărul total de elemente din cele două șiruri trebuie să fie par
d) nici una dintre cele trei variante anterioare

Întrebarea #22:
Dacă se dorește căutarea unui număr într-un șir ordonat de 1000 de elemente, folosind algoritmul căutării binare, atunci numărul maxim al pașilor efectuați va fi:
a) cuprins între 1 și 6
b) cuprins între 7 și 13
c) cuprins între 14 și 20
d) mai mare decât 20.

Întrebarea #23:
Parametrii unui subprogram pot fi transmiși prin:
a) valoare formală sau valoare efectivă
b) referință formală sau referință efectivă
c) valoare sau referință
d) numai prin referință formală

Întrebarea #24:
Care dintre următoarele afirmații este adevărată?
a) un subprogram poate avea cel mult 12 parametri
b) un subprogram trebuie să conțină cel puțin o instrucțiune repetitivă
c) un subprogram nu poate fi apelat din cadrul unui alt subprogram
d) nici una dintre cele trei

Întrebarea #25:
Versiune Pascal:
Care dintre declarații este validă pentru a declara o procedură în limbajul Pascal?
a) `procedure proc(var a:byte);`
b) `procedure proc(a:array[1..100] of integer);`
c) `procedure proc(a:byte):byte;`
d) `procedure procedure(a:byte);`

Versiune C:
Care dintre declarații este validă pentru a declara o procedură în limbajul C?

a) în limbajul C nu există proceduri
b) `int proc(char a)`
c) `void proc(void)`
d) `int proc(a:char):char`

Întrebarea #26:

Versiune Pascal:

Ce se va afișa în urma execuției programului:

```
program Valori;
var a,b:integer;

procedure aduna;
var a:integer;
begin
  a:=a+2;
  b:=b+2;
end;
begin
  a:=3;
  b:=3;
  aduna;
  writeln(a,' ',b);
end.
```

a) 5 5 b) 5 3 c) 3 5 d) 3 3

Versiune C:

Ce se va afișa în urma execuției programului:

```
#include <stdio.h>
int a,b;

void aduna(void){
  int a;
  a+=2;
  b+=2;
}
void main(void){
  a = b = 3;
  aduna();
  printf("%d %d\n",a,b);
}
```

a) 5 5 b) 5 3 c) 3 5 d) 3 3

Întrebarea #27:

Recursivitatea poate fi:

a) directă sau indirectă b) interactivă sau pasivă
c) completă sau incompletă d) formală sau efectivă

Întrebarea #28:

Versiune Pascal:

Ce valoare returnează următoarea funcție:

```
function f(n:integer);
begin
  if n>0 then f:=n*f(n-1);
  else f:=0;
end.
```

a) n! b) (n-1)!
c) 0 d) se va intra în ciclu infinit

Versiune C:

Ce valoare returnează următoarea funcție:

```
int f(int n){
  return (n)?n*f(n-1):0;
}
```

a) n! b) (n-1)!
c) 0 d) se va intra în ciclu infinit

Întrebarea #29:

În care dintre următoarele situații este recomandată folosirea metodei *backtracking*:

a) când problema este definită recursiv
b) când trebuie determinate toate soluțiile unei probleme
c) când trebuie găsit cel mai rapid algoritm
d) nici una dintre cele trei.

Întrebarea #30:

Versiune Pascal:

Ce realizează următorul program:

```
program Backtracking;
var a:array[0..100] of integer;
    m,n:integer;

procedure ScrieSolutie;
var i:integer;
begin
  write(a[1]);
  for i:=2 to m do
    write(' ',a[i]);
  writeln;
end;

procedure Back(pas:integer);
var i:integer;
begin
  if pas = m + 1 then
    ScrieSolutie
  else
    for i:=a[pas-1]+1 to n do
    begin
      a[pas]:=i;
      Back(pas+1);
    end;
end;

begin
  readln(n,m);
  a[0]:=0;
  Back(1);
end.
```

a) determină toate partițiile numărului n formate din m termeni
b) determină toate submulțimile mulțimii {1, 2, ..., m}
c) determină permutările mulțimii {1, 2, ..., n}
d) determină toate submulțimile mulțimii {1, 2, ..., n} care au m elemente

Versiune C:

Ce realizează următorul program:

```
#include <stdio.h>
int m,n,a[101];

void ScrieSolutie(void){
  int i;
  printf("%d",a[1]);
  for(i=2;i<=m;i++)
```

```
    printf(" %d",a[i]);
  putchar('\n');
}
void Back(int pas){
  int i;
  if (pas == m + 1)
    ScrieSolutie();
  else
    for(i=a[pas-1]+1;i<=n;i++){
      a[pas] = i;
      Back(pas+1);
    }
}

void main(void){
  scanf("%d%d",&n,&m);
  a[0] = 0;
  Back(1);
}
```

a) determină toate partițiile numărului n formate din m termeni
b) determină toate submulțimile mulțimii {1, 2, ..., m}
c) determină permutările mulțimii {1, 2, ..., n}
d) determină toate submulțimile mulțimii {1, 2, ..., n} care au m elemente

Întrebarea #31:

O tablă de șah poate fi privită ca fiind o matrice cu opt linii și opt coloane. Două turnuri se atacă unul pe altul dacă se află pe aceeași linie sau pe aceeași coloană. Problema determinării tuturor posibilităților de amplasare a opt turnuri pe o tablă de șah astfel încât oricare două dintre ele să nu se atace este echivalentă cu:
a) problema celor opt regine (dame)
b) problema determinării tuturor permutărilor mulțimii {1, 2, 3, 4, 5, 6, 7, 8}
c) problema determinării tuturor submulțimilor mulțimii {1, 2, 3, 4, 5, 6, 7, 8}
d) nici una dintre cele trei

Întrebarea #32:

Versiune Pascal:

Următorul program ar trebui să determine toate submulțimile mulțimii {1, 2, ..., n}. El conține o singură eroare. Precizați numărul liniei pe care se află eroarea.

```
 1: program Submultimi;
 2: var a:array[0..100] of integer;
 3:     n:integer;

 4: procedure ScrieSolutie(pas:integer);
 5: var i:integer;
 6: begin
 7:   write('{');
 8:   if pas>=1 then
 9:     write(a[1]);
10:   for i:=2 to pas do
11:     write(',',a[i]);
12:   writeln('}')
13: end;

14: procedure Sub(pas:integer);
15: var i:integer;
16: begin
17:   ScrieSolutie(pas-1);
18:   for i:=a[pas-1]+1 to n do
19:   begin
20:     a[pas]:=i;
21:     if pas<n then
22:       Sub(pas+1)
23:   end
24: end;

25: begin
26:   readln(n);
27:   a[0]:=0;
28:   Sub(1)
29: end.
```

a) 8 b) 17 c) 21 d) 22

Versiune C:

Următorul program ar trebui să determine toate submulțimile mulțimii {1, 2, ..., n}. El conține o singură eroare. Precizați numărul liniei pe care se află eroarea.

```
 1: #include <stdio.h>
 2: int n,a[101];

 3: void ScrieSolutie(int pas){
 4:   int i;
 5:   putchar('{');
 6:   if (pas>=1)
 7:     printf("%d",a[1]);
 8:   for (i=2;i<=pas;i++)
 9:     printf(",%d",a[i]);
10:   puts("}");
11: }

12: void Back(int pas){
13:   int i;
14:   ScrieSolutie(pas-1);
15:   for (i=a[pas-1]+1;i<=n;i++){
16:     a[pas] = i;
17:     if (pas < n)
18:       Back(pas+1);
19:   }
20: }

21: void main(void){
22:   scanf("%d",&n);
23:   a[0] = 0;
24:  Back(1);
25: }
```

a) 6 b) 14 c) 17 d) 18

Întrebarea #33:

Versiune Pascal:

Ce se va afișa pe ecran în urma executării următorului program?

```
program Backtracking;
var a:array[0..100] of integer;
    n:integer;

procedure ScrieSolutie;
var i:integer;
begin
  write(a[1]);
  for i:=2 to n do
    write(' ',a[i]);
  writeln;
end;
```

```
procedure Back(pas:integer);
var i:integer;
begin
  for i:=1 to n do
  begin
    a[pas]:=i;
    if pas>0 then
      Back(pas-1)
    else
      if pas = n + 1 then
        ScrieSolutie;
  end
end;

begin
  readln(n);
  a[0]:=0;
  Back(n)
end.
```

a) nimic
b) permutările unei mulțimi
c) submulțimile unei mulțimi
d) elementele unui produs cartezian

Versiune C:
Ce se va afișa pe ecran în urma executării următorului program?

```
#include <stdio.h>
int n,a[101];

void ScrieSolutie(){
  int i;
  printf("%d",a[1]);
  for (i=2;i<=n;i++)
    printf(" %d",a[i]);
  putchar('\n');
}

void Back(int pas){
  int i;
  for (i=1;i<=n;i++){
    a[pas] = i;
    if (pas > 0)
      Back(pas-1);
    else
      if (pas == n + 1)
        ScrieSolutie();
  }
}

void main(void){
  scanf("%d",&n);
  a[0] = 0;
  Back(n);
}
```

a) nimic
b) permutările unei mulțimi
c) submulțimile unei mulțimi
d) elementele unui produs cartezian

Întrebarea #34:
Se consideră o mulțime al cărei cardinal este 6. Aceasta va avea a permutări, b submulțimi, dintre care c submulțimi vor avea cel puțin 3 elemente. Valorile a, b și c sunt:
a) a = 720, b = 32, c = 16
b) a = 720, b = 64, c = 42
c) a = 720, b = 64, c = 32
d) a = 120, b = 32, c = 16

Întrebarea #35:
Care dintre următoarele afirmații este falsă:
a) din stive elementele sunt eliminate în ordinea inversă inserării lor
b) din cozi elementele sunt eliminate în ordinea inserării lor
c) stivele pot fi utilizate pentru a simula recursivitatea directă
d) cozile pot fi utilizate pentru a simula recursivitatea indirectă

Întrebarea #36:
Pentru a insera un element într-o listă dublu înlănțuită ordonată trebuie modificată(e):
a) două referințe
b) patru referințe
c) o referință
d) nici o referință.

Întrebarea #37:
Se consideră o stivă inițial vidă. Care este suma valorilor afișate în urma execuției următoarelor operații:

```
Adaugă_în_stivă(1);
Adaugă_în_stivă(2);
Elimină_primul_element_al_stivei;
Adaugă_în_stivă(3);
Adaugă_în_stivă(4);
Scrie_primul_element_al_stivei;
Scrie_primul_element_al_stivei;
Adaugă_în_stivă(5);
Adaugă_în_stivă(6);
Elimină_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
Adaugă_în_stivă(7);
Adaugă_în_stivă(8);
Scrie_primul_element_al_stivei;
Adaugă_în_stivă(9);
Elimină_primul_element_al_stivei;
Scrie_primul_element_al_stivei;
Adaugă_în_stivă(10);
Elimină_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
Scrie_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
Scrie_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
Scrie_primul_element_al_stivei;
Elimină_primul_element_al_stivei;
```

a) 35 b) 37 c) 39 d) 41

Întrebarea #38:
Ce se înțelege prin santinelă, în cazul structurilor dinamice de date?
a) un element fictiv care nu face parte din structura propriu-zisă și care este utilizat pentru simplificarea implementării operațiilor de inserare și eliminare
b) un element al structurii care conține primul element din structură potrivit unui anumit criteriu de ordonare

c) primul element al structurii
d) nici una dintre cele trei

Întrebarea #39:
Care dintre următoarele afirmații este falsă?
a) un graf neorientat trebuie să conțină cel puțin o muchie
b) un graf neorientat poate avea mai multe vârfuri izolate
c) o muchie poate fi privită ca fiind o pereche de noduri
d) lungimea unui ciclu este dată de numărul muchiilor care formează ciclul respectiv

Întrebarea #40:
Care dintre următoarele afirmații este adevărată?
a) un subgraf al unui graf G are întotdeauna același număr de noduri ca și graful G
b) un graf conex este întotdeauna și un graf eulerian
c) un graf complet este întotdeauna și un graf regulat
d) un graf hamiltonian este întotdeauna și un graf regulat

Întrebarea #41:
Care este gradul maxim al unui nod din graful neorientat descris prin următoarea matrice de adiacență?

```
0 1 0 1 0 0 0 1 1 1
1 0 1 1 1 1 1 0 1 1
0 1 0 0 1 0 0 1 1 0
1 1 0 0 0 1 0 1 1 0
0 1 1 0 0 1 1 1 0 1
0 1 0 1 1 0 1 0 0 0
0 1 0 0 1 1 0 0 0 1
1 0 1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0
1 1 0 0 1 0 1 0 0 0
```

a) 6 b) 7 c) 8 d) 9

Întrebarea #42:
Într-un graf orientat, gradul unui nod este dat de:
a) maximul dintre gradul interior și gradul exterior
b) minimul dintre gradul interior și gradul exterior
c) suma dintre gradul interior și gradul exterior
d) nici una dintre cele trei

Grila

1. a.☐ b.☐ c.☐ d.☐	16. a.☐ b.☐ c.☐ d.☐	31. a.☐ b.☐ c.☐ d.☐
2. a.☐ b.☐ c.☐ d.☐	17. a.☐ b.☐ c.☐ d.☐	32. a.☐ b.☐ c.☐ d.☐
3. a.☐ b.☐ c.☐ d.☐	18. a.☐ b.☐ c.☐ d.☐	33. a.☐ b.☐ c.☐ d.☐
4. a.☐ b.☐ c.☐ d.☐	19. a.☐ b.☐ c.☐ d.☐	34. a.☐ b.☐ c.☐ d.☐
5. a.☐ b.☐ c.☐ d.☐	20. a.☐ b.☐ c.☐ d.☐	35. a.☐ b.☐ c.☐ d.☐
6. a.☐ b.☐ c.☐ d.☐	21. a.☐ b.☐ c.☐ d.☐	36. a.☐ b.☐ c.☐ d.☐
7. a.☐ b.☐ c.☐ d.☐	22. a.☐ b.☐ c.☐ d.☐	37. a.☐ b.☐ c.☐ d.☐
8. a.☐ b.☐ c.☐ d.☐	23. a.☐ b.☐ c.☐ d.☐	38. a.☐ b.☐ c.☐ d.☐
9. a.☐ b.☐ c.☐ d.☐	24. a.☐ b.☐ c.☐ d.☐	39. a.☐ b.☐ c.☐ d.☐
10. a.☐ b.☐ c.☐ d.☐	25. a.☐ b.☐ c.☐ d.☐	40. a.☐ b.☐ c.☐ d.☐
11. a.☐ b.☐ c.☐ d.☐	26. a.☐ b.☐ c.☐ d.☐	41. a.☐ b.☐ c.☐ d.☐
12. a.☐ b.☐ c.☐ d.☐	27. a.☐ b.☐ c.☐ d.☐	42. a.☐ b.☐ c.☐ d.☐
13. a.☐ b.☐ c.☐ d.☐	28. a.☐ b.☐ c.☐ d.☐	43. a.☐ b.☐ c.☐ d.☐
14. a.☐ b.☐ c.☐ d.☐	29. a.☐ b.☐ c.☐ d.☐	44. a.☐ b.☐ c.☐ d.☐
15. a.☐ b.☐ c.☐ d.☐	30. a.☐ b.☐ c.☐ d.☐	45. a.☐ b.☐ c.☐ d.☐

Întrebarea #43:
Care dintre următoarele corespondențe nu este corectă:
a) noțiunii de muchie în grafurile neorientate îi corespunde noțiunea de arc în grafurile orientate
b) noțiunii de ciclu în grafurile neorientate îi corespunde noțiunea de circuit în grafurile orientate
c) noțiunii de lanț în grafurile neorientate îi corespunde noțiunea de drum în grafurile orientate
d) noțiunii de adiacență în grafurile neorientate îi corespunde noțiunea de incidență în grafurile orientate

Întrebarea #42:
Care dintre următoarele afirmații este falsă?
a) o frunză într-un arbore cu rădăcină are întotdeauna gradul 1
b) singurul nod dintr-un arbore cu rădăcină care nu are părinte este rădăcina
c) orice nod al unui arbore cu rădăcină poate avea cel mult un frate
d) nici una dintre cele trei

Întrebarea #45:
Se consideră un arbore a cărui rădăcină este nodul 1 și care conține alte 9 noduri identificate prin numere cuprinse între 2 și 10. Arborele conține muchii între următoarele perechi de noduri: (1, 2), (1, 5), (2, 7), (3, 5), (3, 10), (4, 5), (6, 10), (7, 9) și (8, 10). Stabiliți șirul părinților celor nouă noduri (identificate prin numere cuprinse între 2 și 10), în ordinea dată de numerele de identificare ale acestor noduri. Așadar, primul element al șirului va indica numărul de ordine al părintelui nodului 2, al doilea element va indica numărul de ordine al părintelui nodului 3 etc.
a) 1 5 5 1 10 2 10 7 3
b) 1 5 5 3 10 2 10 7 6
c) 1 2 3 4 5 6 7 8 9
d) 9 8 7 6 5 4 3 2 1

Soluții

Grila completată corect pentru acest test, împreună cu explicațiile referitoare la răspunsurile corecte, va fi publicată în următorul număr al revistei noastre.

Cei care doresc să afle mai repede răspunsurile corecte, pot să decupeze grila alăturată completată cu răspunsurile și să o trimită pe adresa redacției. În scrisoare trebuie indicată adresa *e-mail* la care se dorește primirea răspunsurilor corecte.

Observații

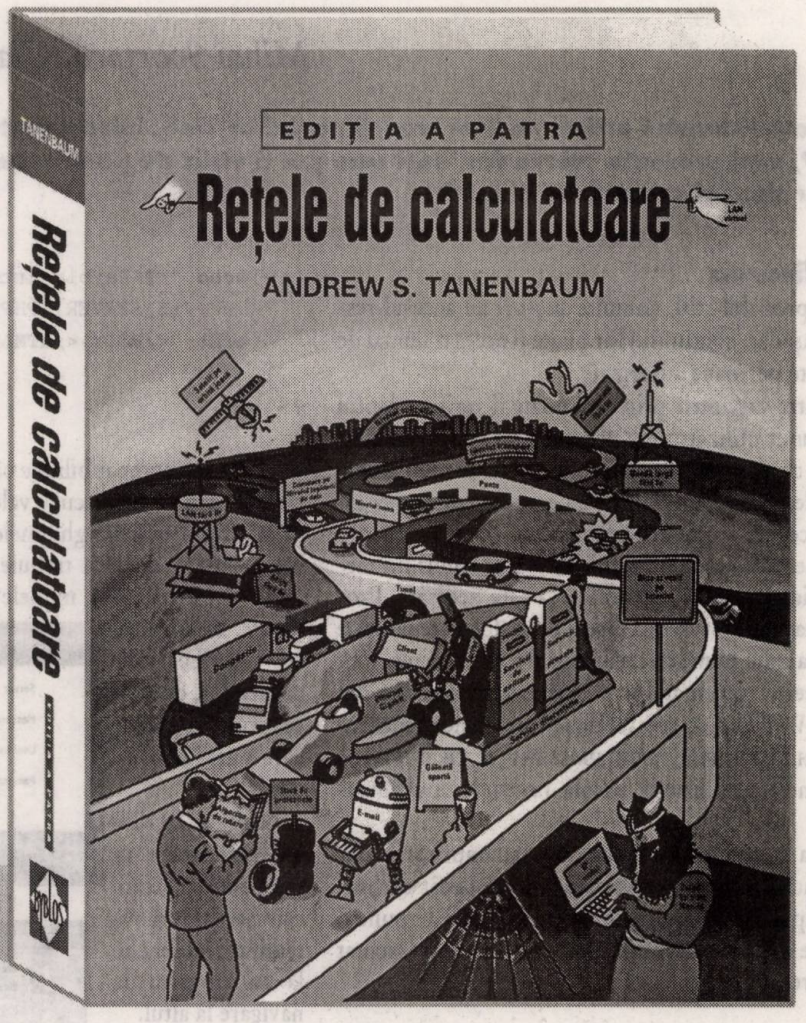
Caracterul acestui test este orientativ. Acest test este neoficial, el ilustrând doar concepția noastră referitoare la modul în care ar putea să arate o astfel de grilă. Este foarte posibil ca grila de la *Bacalaureat* să fie complet diferită.

Redacția *GInfo* nu își asumă nici o răspundere privitoare la relevanța și corectitudinea acestui test.

Grila constă în **45** de întrebări, pentru fiecare existând ***patru*** variante de răspuns dintre care ***exact una*** este corectă.

Cine are carte, are parte!

... de înțelegerea rețelelor!



Cea mai bună introducere din lume în rețele de calculatoare
- complet actualizată pentru tehnologiile cheie de mâine -

Comand-o acum și dacă ești student primești și o reducere semnificativă!

Byblos srl, tel: 021-330.92.81, fax: 021-330.92.85, e-mail: dpastrama@byblos.ro
www.byblos.ro
Librăria Eminescu, Universitate, la subsol, tel: 021-312.72.65; 0721-927.499

Hello World! (20)

Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

În cadrul acestui articol dedicat creării paginilor Web, folosind interpretorul PHP, vom prezenta câteva facilități care pot fi utilizate pentru construirea unor site-uri cu conținut dinamic.

Autentificarea

După cum probabil știți, anumite *site*-uri au accesul restricționat. Așadar, conținutul lor poate fi accesat numai de către anumite persoane autorizate.

Una dintre variantele prin care poate fi verificat faptul dacă un anumit utilizator are dreptul de a accesa un anumit *site* este cererea introducerii unui nume de utilizator și a unei parole în momentul în care persoana respectivă încearcă să acceseze aceste informații.

În momentul accesării paginii *Web* este trimis un mesaj care indică faptul că autentificarea este necesară. Programul de navigare va afișa o casetă de dialog care permite introducerea unui nume de utilizator și a unei parole. După introducerea informațiilor necesare va fi accesată din nou pagina *Web* dar, de această dată, variabilele predefinite `PHP_AUTH_USER`, `PHP_AUTH_PW` și `PHP_AUTH_TYPE` vor conține numele de utilizator, parola, respectiv tipul autentificării.

Pentru a trimite mesajul care cere autentificarea este utilizată funcția `header()`. Nu este posibilă decât autentificarea de tip "***Basic***". Un document *PHP* care impune o autentificare înaintea accesării informațiilor este prezentat în continuare:

```
<?PHP
  if (!isset($_SERVER['PHP_AUTH_USER'])){
    header('WWW-Authenticate: Basic
                              realm="GInfo"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Pentru accesarea conținutului este
                necesară o autentificare!';
    exit;
  }
  else{
    echo "<HTML><HEAD>";
    echo "<TITLE>Autentificare</TITLE>";
    echo "</HEAD><BODY>";
    echo "<P>Salut <B><I>
        {$_SERVER['PHP_AUTH_USER']}</I></B>!";
    echo "<P>Parola introdusă este <B><I>
        {$_SERVER['PHP_AUTH_PW']}</I></B>.";
    echo "</BODY></HTML>";
  }
?>
```

Pentru o compatibilitate cât mai mare, cuvântul `Basic` trebuie scris cu majusculă, valoarea pentru `realm` trebuie să fie cuprinsă între ghilimele (nu între apostrofuri), iar între `HTTP/1.0` și `401` trebuie să existe exact un spațiu.

În acest exemplu, numele de utilizator și parola sunt afișate. În practică, se va verifica validitatea lor, de obicei printr-o interogare a unei baze de date.

Figura 1: Opera 7.02

Caseta de dialog care apare în momentul în care este necesară o autentificare diferă de la un program de navigare la altul.

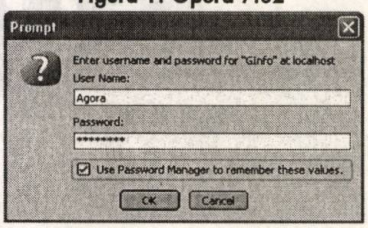


Figura 2: Netscape Navigator 7.02

În figurile 1, 2 și 3 sunt prezentate casetele de dialog afișate de ultimele versiuni ale celor mai cunoscute *browser*-e: *Opera* 7.02, *Netscape Navigator* 7.02 și *Microsoft Internet Explorer* 6.0 *Service Pack* 1.

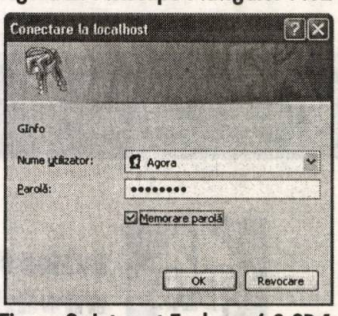


Figura 3: Internet Explorer 6.0 SP 1

În cazul în care este instrodus un nume de utilizator

și o parolă și este apăsat butonul *Ok*, aceste informații vor fi afișate. Pagina *Web* generată este prezentată în figura 4.

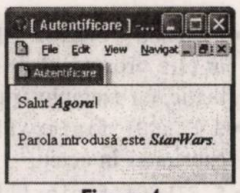


Figura 4

În situația în care este apăsat butonul *Cancel* (sau cel echivalent; de exemplu, *Revocare* în *Internet Explorer* 6 localizat pentru limba română) pagina *Web* generată va conține un mesaj de avertizare; situația este ilustrată în figura 5.

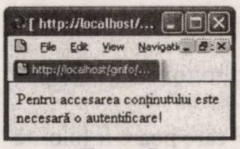


Figura 5

Cookie-uri

În cadrul unui articol dedicat limbajului *JavaScript* am prezentat modul în care pot fi stocate informații pe calculatorul care accesează o anumită pagină *Web* și cum pot fi acestea accesate ulterior.

În *PHP* există o funcție specială care poate fi folosită pentru crearea *cookie*-urilor și anume `setcookie()`. Funcția are șase parametri, doar prezența primului fiind obligatorie:

- `nume`: șir de caractere care reprezintă numele *cookie*-ului;
- `valoare`: șir de caractere care reprezintă valoarea *cookie*-ului (informația corespunzătoare);
- `expirare`: un număr întreg care indică momentul de timp în care *cookie*-ul nu va mai putea fi utilizat;
- `cale`: șir de caractere care reprezintă o cale generică a fișierelor care pot accesa *cookie*-ul;
- `domeniu`: șir de caractere care reprezintă domeniul pe care trebuie stocate fișierele care pot accesa *cookie*-ul;
- `securitate`: număr întreg care, dacă are valoarea `1`, arată că informația poate fi transmisă doar folosindu-se o conexiune *HTTPS* securizată.

Dacă unul dintre parametri este prezent, atunci toți parametrii anteriori trebuie să fie prezenți. Dacă unul dintre aceștia nu este utilizat, se va folosi în poziția respectivă fie șirul vid, fie valoarea 0.

În continuare vom prezeta câteva exemple de apeluri prin care sunt create *cookie*-uri:

```
setcookie("TestCookie","GInfo");
setcookie("TestCookie","GInfo",time()+3600);
            // expiră după o oră de la creare
setcookie("TestCookie","GInfo",time()+3600,
                    "/revista/",".ginfo.ro",1);
```

Cu ajutorul funcției `setcookie()` este posibilă și ștergerea *cookie*-urilor. Următoarele exemple ilustrează modul în care poate fi realizată această operație:

```
setcookie("TestCookie","",time()-3600);
setcookie("TestCookie","",time()-3600,
                "/concurs/", ".ginfo.ro", 1);
```

Se observă că momentul expirării este setat ca fiind anterior momentului executării operației de ștergere.

Există posibilitatea de a crea șiruri de *cookie*-uri prin simpla includere a parantezelor drepte în denumiri. Un exemplu de creare și utilizare a unor astfel de șiruri este prezentat în continuare:

```
setcookie("cookie[three]","cookiethree");
setcookie("cookie[two]","cookietwo");
setcookie("cookie[one]","cookieone");
if (isset($cookie)) {
  while (list($name,$value) = each($cookie)){
    echo "$name == $value<BR>";
  }
}
```

Pentru a accesa *cookie*-urile stocate pe calculatorul unui vizitator al paginii *Web*, este disponibil vectorul `$_COOKIE`. Vom prezenta în continuare un document *PHP* cu ajutorul căruia se generează o pagină care conține denumirile și valorile tuturor *cookie*-urilor.

```
<HTML>
  <HEAD>
    <TITLE>
      Cookie-uri
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      foreach ($_COOKIE as $cheie => $valoare)
        echo "$cheie = $valoare<BR>";
    ?>
  </BODY>
</HTML>
```

Încărcarea fișierelor

Fișierele pot fi încărcate pe un *server* dacă utilizatorul folosește un program de navigare compatibil *RFC*-1867; pot fi utilizate ultimele versiuni ale celor mai cunoscute *browser*-e.

Este posibilă încărcarea atât a fișierelor text, cât și a celor binare. Folosind autentificarea *PHP* și funcțiile de manipulare a fișierelor, avem control total asupra utilizatorilor care pot încărca fișiere și asupra operațiilor efectuate cu fișierul încărcat.

Formularul de încărcare

Pentru ca utilizatorul să poată încărca un fișier, el trebuie să aibă la dispoziție un formular care să permită alegerea fișierului.

Un astfel de formular poate fi creat, de exemplu, folosind limbajul *HTML* standard. Prezentăm în continuare un document *HTML* care generează o pagină *Web* care permite alegerea unui fișier și încărcarea sa pe un *server*.

Din nou formularul diferă în funcție de programul de navigare utilizat. În figurile 6, 7 și 8 sunt prezentate formularele afișate de programele de navigare *Opera* 7.02, *Netscape Navigator* 7.02 și *Internet Explorer* 6.0 *SP* 1.

```
<HTML>
  <HEAD>
    <TITLE>
      Încărcarea unui fişier
    </TITLE>
  </HEAD>
  <BODY>
    <FORM enctype=
      "multipart/form-
      data" action =
        "http://www.ginfo.
        ro/php/upload.php"
      method = "post">
      Alegeţi fişierul:
      <BR>
      <INPUT name =
                  "fisier"
        type = "file">
      <INPUT type =
        "hidden" name =
        "MAX_FILE_SIZE"
          value = 2457>
      <P>
      <INPUT type = "submit" value =
                          "Trimite fişierul!">
    </FORM>
  </BODY>
</HTML>
```

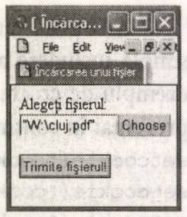


Figura 6: Opera

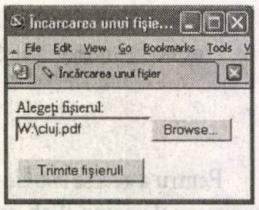


Figura 7: Netscape

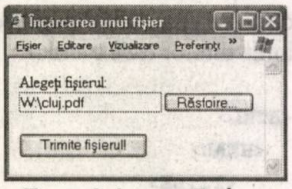


Figura 8: Internet Explorer

Butonul *Browse...* (*Choose*, *Răsfoire...* etc.) poate fi utilizat pentru alegerea unui fişier care urmează a fi încărcat. Apăsarea acestui buton va duce la apariţia unei casete de dialog care permite alegerea fişierului.

Este posibilă specificarea fişierului precizând exact calea în caseta de text afişată.

URL-ul care este folosit ca valoare pentru atributul **action** al marcajului **<FORM>** trebuie să reprezinte documentul *PHP* care va realiza încărcarea.

Încărcarea fişierului va începe în momentul în care este apăsat butonul *Trimite fişierul!*.

Operaţia de încărcare

Aşadar, formularul prin intermediul căruia este ales fişierul va fi prelucrat de către un *script PHP*.

Pentru a manipula fişierele poate fi utilizat tabloul `$_FILES`. Pentru *script*-ul care prelucrează formularul prezentat anterior, informaţiile referitoare la fişierul încărcat vor fi accesate folosind variabila `$_FILES['fisier']` care este tot un vector.

Elementele acestui vector sunt prezentate în cele ce urmează:

- `$_FILES['fisier']['name']`: numele pe care îl are fişierul pe calculatorul utilizatorului;
- `$_FILES['fisier']['tmp_name']`: numele temporar pe care îl are fişierul pe *server* după încheierea procesului de încărcare;
- `$_FILES['fisier']['type']`: tipul fişierului în cazul în care programul de navigare furnizează această informaţie; un exemplu ar putea fi `image/gif`;
- `$_FILES['fisier']['size']`: dimensiunea fişierului, exprimată în octeţi;
- `$_FILES['fisier']['error']`: un cod de eroare generat în urma efectuării încărcării.

Fişierul va fi stocat pe *server* în directorul care conţine fişierele temporare.

O secvenţă de instrucţiuni *PHP* care poate fi folosită pentru a prelua fişierul încărcat şi a-l muta în directorul dorit este următoarea:

```
<?PHP
  if (is_uploaded_file($_FILES['fisier']
                                ['tmp_name'])) {
    copy($_FILES['fisier']['tmp_name'],
                          "/directorul/dorit");
  }
  else{
    echo "Eroare de transfer " .
                      $_FILES['fisier']['name'];
  }
?>
```

O variantă alternativă este:

```
<?PHP
  move_uploaded_file($_FILES['fisier']
      ['tmp_name'],"/directorul/dorit");
?>
```

Coduri de eroare

Începând cu versiunea *PHP* 4.2.0, în urma efectuării unui transfer, sunt generate anumite coduri de eroare. Versiunea 4.3.0 introduce constante predefinite pentru fiecare dintre aceste coduri.

În funcţie de codul de eroare generat în urma operaţiei, trebuie realizate diferite acţiuni. Codul de eroare poate fi accesat prin intermediul elementului `['error']` al vectorului care păstrează informaţiile despre fişier.

Codurile de eroare care pot fi generate sunt următoarele:

- `UPLOAD_ERR_OK` (0): nu a fost semnalată nici o eroare în timpul transferului; operaţia de încărcare s-a încheiat cu succes;
- `UPLOAD_ERR_INI` (1): dimensiunea fişierului este mai mare decât valoarea maximă permisă (această valoare este specificată în momentul configurării *server*-ului şi poate fi schimbată modificând fişierul **`PHP.INI`**);
- `UPLOAD_ERR_FORM_SIZE` (2): dimensiunea fişierului este mai mare decât valoarea maximă indicată de directiva `MAX_FILE_SIZE` care este specificată în formular; această valoare este specificată prin intermediul unui câmp ascuns al cărui nume este `MAX_FILE_SIZE`;

- UPLOAD_ERR_PARTIAL (3): fișierul a fost încărcat doar parțial;
- UPLOAD_ERR_NO_FILE (4): nu a fost încărcat nici un fișier.

Încărcarea mai multor fișiere

Există posibilitatea de a încărca mai multe fișiere dacă sunt folosite nume diferite pentru marcajul **<INPUT>** utilizat pentru afișarea casetei de text și a butonului folosite pentru alegerea fișierului.

De asemenea, este posibil ca fișierele încărcate să fie organizate automat într-un vector. Pentru aceasta, la numele folosit trebuie adăugate caracterele `'[]'`. Un exemplu este prezentat în continuare:

```
<FORM action =
          "http://www.ginfo.ro/php/upload.php"
  method = "post"
  enctype = "multipart/form-data">
  Alegeți fișiele:<BR>
  <INPUT name = "fisiere[]" type = "file">
  <BR>
  <INPUT name = "fisiere[]" type = "file">
  <BR>
  <INPUT type = "submit" value =
                    "Trimite fișierele">
</FORM>
```

Să presupunem că am ales să încărcăm fișierele **F1.PDF** și **F2.PDF**. În acest caz, `$_FILES['fisiere']['name'][0]` va avea valoarea `F1.PDF`, în timp ce `$_FILES['fisiere']['name'][1]` va avea valoarea `F2.PDF`.

Dimensiunile celor două fișiere pot fi determinate folosind valorile `$_FILES['fisiere']['size'][0]`, respectiv `$_FILES['fisiere']['size'][1]`.

Includerea fișierelor

Deseori, mai multe pagini *Web* pe care le creăm trebuie să conțină secvențe identice. Evident, putem copia secvențele dintr-un fișier în altul, dar această variantă, pe lângă faptul că este incomodă, duce la un consum mărit de spațiu pe *server* și, în cazul în care secvențele sunt lungi sau sunt folosite în foarte multe pagini, problema este destul de importantă.

Interpretorul *PHP* permite păstrarea secvențelor comune în fișiere separate și accesarea acestora din cadrul altor documente *PHP*.

Pentru a accesa un astfel de fișier, poate fi utilizată directiva **include**. Practic, din punct de vedere logic, tot conținutul fișierului inclus este inserat în poziția în care apare directiva. Sintaxa acestei directive este **include** *fișier*;.

Să presupunem că avem un fișier numit **HELLO.TXT** cu următorul conținut:

```
<?PHP
  echo "Hello World!";
?>
```

În continuare vă prezentăm modul în care poate fi inclus acest fișier într-un document *PHP*. Pagina *Web* generată este prezentată în figura 9.

```
<HTML>
  <HEAD>
    <TITLE>
      Fișiere incluse
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      include "HELLO.TXT";
    ?>
  </BODY>
</HTML>
```

Figura 9

Pentru ca fișierul inclus să fie interpretat, acesta trebuie obligatoriu să indice faptul că secvențele de text din cadrul său reprezintă instrucțiuni *PHP*. Așadar, prezența construcției **<?PHP ... ?>** este necesară pentru ca textul să fie interpretat corect. Dacă fișierul **HELLO.TXT** ar conține doar secvența **echo** `"Hello World!";`, atunci aceasta nu ar fi interpretată ca fiind cod *PHP*, iar efectul ar fi cel din figura 10.

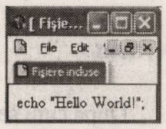


Figura 10

Aceasta se datorează faptului că în momentul includerii unui fișier se iese din modul *PHP* și tot textul este interpretat ca fiind cod *HTML*. După interpretarea textului din fișierul inclus se intră din nou în modul *PHP*.

În concluzie, folosirea celor două fișiere este echivalentă cu utilizarea următorului fișier:

```
<HTML>
  <HEAD>
    <TITLE>
      Fișiere incluse
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "Hello World!";
    ?>
  </BODY>
</HTML>
```

Includerea în cadrul funcțiilor

Dacă directiva de includere apare în cadrul unei funcții, atunci se consideră că toate instrucțiunile din cadrul fișierului inclus fac parte din corpul funcției. Așadar, domeniul de vizibilitate al variabilelor folosite în fișierul inclus este același cu domeniul pe care l-ar fi avut dacă ar fi apărut în cadrul funcției. Pentru a ilustra acest aspect vom considera un fișier **VAR.TXT** care conține secvența:

```
<?PHP
  $culoare = 'verde';
```

```
  $fruct = 'măr';
?>
```

și următorul document *PHP*:

```
<HTML>
  <HEAD>
    <TITLE>
      Includere în funcție
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      function fructe(){
        global $culoare;
        include 'var.txt';
        echo "un $fruct $culoare<BR>";
      }
      fructe();
      echo "un $fruct $culoare<BR>";
    ?>
  </BODY>
</HTML>
```

După cum se poate vedea în figura 11, variabilele definite în cadrul fișierului inclus au domeniul de vizibilitate corespunzător funcției `fructe()`. Din aceste motive valoarea variabilei `$fruct` nu este disponibilă în afara funcției. Valoarea variabilei `$culoare` este disponibilă și în afara funcției, deoarece a fost declarată ca fiind o variabilă globală.

Figura 11

Așadar, instrucțiunea **echo** din cadrul funcției duce la afișarea mesajului un măr verde, deoarece ambele valori ale variabilelor sunt disponibile, iar instrucțiunea din afara funcției duce la afișarea mesajului un verde, deoarece, în afara funcției, variabila `$fruct` nu a fost definită anterior și este automat inițializată cu șirul vid.

Includeri prin HTTP

După cum ați observat, directiva **include** are ca parametru un șir de caractere care conține numele fișierului care va fi inclus. Acesta poate fi orice *URL* valid, așadar este posibilă includerea unui fișier aflat pe un alt *server*.

Un exemplu ar putea fi **include** `"http/www.ginfo.ro/php/ginfo.txt"`. O astfel de directivă funcționează doar dacă se folosește *PHP* 4.3.0 sau o versiune ulterioară.

Observație

Datorită faptului că secvențele din cadrul fișierului inclus sunt, din punct de vedere logic, inserate în poziția în care apare directiva **include**, trebuie să fim atenți la modul în care sunt folosite blocurile de instrucțiuni.

Să presupunem că avem un fișier **HELLO1.TXT** care conține secvența:

```
<?PHP
  echo "1. ";
  echo "Hello World!<BR>";
?>
```

și un fișier **HELLO2.TXT** cu următorul conținut:

```
<?PHP
  echo "2. ";
  echo "Hello World!<BR>";
?>
```

Dacă în cadrul unui document *PHP* folosim o secvență de tipul:

```
if ($ok)
  include "HELLO1.TXT";
else
  include "HELLO2.TXT";
```

atunci aceasta este echivalentă cu:

```
if ($ok)
  echo "1. ";
  echo "Hello World!<BR>";
else
  echo "2. ";
  echo "Hello World!<BR>";
```

Deși construcția din cadrul documentului *PHP* pare corectă, din cea echivalentă rezultă clar faptul că *script*-ul nu va funcționa așa cum ne-am aștepta.

Pentru a evita acest tip de eroare, construcția corectă este:

```
if ($ok){
  include "HELLO1.TXT";
}
else{
  include "HELLO2.TXT";
}
```

chiar dacă acoladele par inutile. Rolul lor devine evident dacă privim construcția echivalentă rezultată:

```
if ($ok){
  echo "1. ";
  echo "Hello World!<BR>";
}
else{
  echo "2. ";
  echo "Hello World!<BR>";
}
```

Includerea unui fișier o singură dată

Uneori, mai ales în situațiile în care se folosesc variabile pentru numele fișierelor incluse, este posibil ca anumite fișiere să fie incluse de mai multe ori. De obicei, fișierele incluse conțin definiții ale unor funcții sau clase, motiv pentru care este de dorit ca un astfel de fișier să nu fie inclus de mai multe ori.

Pentru ca un fișier să fie inclus o singură dată, se utilizează o directivă alternativă numit `include_once`. După cum rezultă din denumire, un fișier va fi inclus o singură dată. Cu alte cuvinte, în momentul în care se cere includerea unui fișier, se verifică dacă acesta a fost deja inclus și în

această situație directiva nu are nici un efect (fișierul nu este inclus încă o dată).

Alternative

Pe lângă directivele **include** și **include_once** există două alte directive a căror utilizare are un efect aproape echivalent. Acestea sunt **require** (echivalentă cu **include**) și **require_once** (echivalentă cu **include_once**).

Singura diferență dintre **include** și **require** (respectiv dintre **include_once** și **require_once**) este dată de modul în care este tratată situația în care fișierul care se dorește a fi inclus nu există sau nu este disponibil.

În cazul directivelor **include** și **include_once** este generat un *avertisment* și execuția *script*-ului continuă fără ca fișierul să fie inclus. Practic, din punct de vedere logic, este ignorată directiva.

În cazul directivelor **require** și **require_once** este generată o eroare fatală și execuția *script*-ului este întreruptă.

Așadar, în situațiile în care este absolut necesară existența fișierului, pentru a nu apărea efecte bizare, trebuie utilizată directiva **require**, respectiv **require_once**.

Starea conexiunii

În *PHP*, conexiunea poate avea una dintre cele trei stări posibile :

- `NORMAL` (0): starea normală de funcționare;
- `ABORTED` (1): utilizatorul care accesează pagina generată de *script*-ul aflat în execuție s-a deconectat (de obicei prin apăsarea butonului *Stop* al programului de navigare);
- `TIMEOUT` (2): a fost atinsă limita de timp impusă de *PHP*.

Pentru fiecare dintre cele trei stări există un *indicator* (*flag*) care este setat sau nu în funcție de starea conexiunii.

În funcție de *script*, este necesar sau nu ca execuția acestuia să se încheie în momentul în care este întreruptă conexiunea. De exemplu, în cazul în care *script*-ul actualizează o bază de date ar fi de dorit ca execuția sa să continue. În principiu, este indicat ca *script*-urile să își încheie execuția chiar dacă datele de ieșire nu vor mai fi trimise utilizatorului.

Pentru a decide dacă execuția *script*-ului se va încheia în momentul întreruperii conexiunii puteți (sau nu!) să apelați funcția `ignore_user_abort()`. Implicit, în momentul în care conexiunea este întreruptă, se întrerupe și execuția *script*-ului.

Este posibil (chiar recomandabil!) să fie definită o funcție care va fi executată înaintea întreruperii execuției *script*-ului. Aceasta poartă denumirea de ***funcție de dezactivare*** (*shutdown function*) și este definită la fel ca orice altă funcție.

Pentru a specifica faptul că o anumită funcție trebuie apelată înaintea întreruperii execuției, aceasta trebuie înregistrată ca fiind funcție de dezactivare. Aceasta se realizează printr-un apel al funcției `register_shutdown_function()`. În cadrul funcției se poate testa dacă execuția a fost determinată de întreruperea conexiunii prin apelarea funcției `connection_aborted()`. Aceasta va returna valoarea **TRUE** dacă a fost întreruptă conexiunea.

Funcția de dezactivare va fi apelată automat și în cazul în care este depășită limita de timp. În cadrul funcției se poate testa dacă un astfel de eveniment a avut loc prin apelarea funcției `connection_aborted()` care returnează valoarea **TRUE** în cazul depășirii acestei limite.

Există posibilitatea de a testa ambele indicatoare printr-un singur apel al funcției `connection_status()`. Funcția va returna un număr întreg ai cărui biți reprezintă indicatoarele. De exemplu, dacă sunt setate indicatoarele `ABORTED` și `TIMEOUT`, valoarea returnată va fi 3.

Trebuie remarcat faptul că este posibil ca indicatorii `ABORTED` și `TIMEOUT` să fie ambii setați la un moment dat. Această situație poate apărea în cazul în care întreruperea conexiunii este ignorată. *PHP* va detecta faptul că a fost întreruptă conexiunea și va seta indicatorul corespunzător. Totuși, execuția *script*-ului va continua și este posibil să expire și limita de timp, moment în care este setat și indicatorul `TIMEOUT`.

Implicit, limita de timp este de 30 de secunde, dar ea poate fi modificată folosind funcția `set_time_limit()`.

Fișiere aflate la distanță

Așa cum am afirmat în secțiunea dedicată includerii fișierelor, începând cu versiunea *PHP* 4.3.0 este posibilă includerea de fișiere aflate pe alte *server*-e.

Evident, este posibilă utilizarea de fișiere aflate pe alte *server*-e și în alte scopuri. Practic, în aproape toate funcțiile care manipulează fișiere, pot fi folosite atât fișiere locale, cât și fișiere aflate la distanță.

În continuare vom prezenta modul în care pot fi citite date dintr-un fișier aflat pe *site*-ul revistei noastre:

```
<?PHP
  $file = fopen("http://www.ginfo.ro/
                              index.shtml","r");
  if (!$file) {
    echo "Fișierul nu poate fi accesat!\n";
    exit;
  }
  ... // citirea datelor
  fclose($file);
?>
```

Este posibilă și scrierea de date pe un *server* aflat la distanță. Fișierul va fi deschis folosind o secvență de tipul:

```
$file = fopen("http://www.ginfo.ro/
                              index.shtml","w");
if (!$file) {
  echo "Accesul nu este permis!\n";
  exit;
}
```

Mihai Scorțaru este redactor-șef GInfo și poate fi contactat prin e-mail la adresa **skortzy@yahoo.com**. *Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa* **csoroiu@yahoo.com**.

COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui episod al serialului dedicat compresiilor de date vă vom prezenta o serie de transformări care au ca scop creșterea performanțelor diferiților algoritmi de compresie.

De-a lungul istoriei compresiilor de date s-a pus întrebarea: "*Există vreo posibilitate de a face ca datele să devină mai comprimabile?*" Ei bine, răspunsul la această întrebare s-a dovedit a fi "*Da, există!*".

În paginile următoare vă vom prezenta câteva transformări care se pot efectua asupra datelor pentru ca acestea să devină *mai comprimabile*, dar mai întâi vom prezenta câteva noțiuni teoretice.

Pentru un algoritm de compresie entropic, un șir de simboluri generat de o sursă de informație *S* se comprimă mai bine (devine *mai comprimabil*) dacă entropia acestuia scade în urma efectuării unei transformări.

Pentru ca o transformare să fie performantă, aceasta trebuie să îndeplinească următoarele condiții:

- ordinul de complexitate al algoritmului de transformare să fie cât mai mic posibil;
- diferența dintre lungimea șirului de simboluri obținut după aplicarea transformării să fie cât mai mică posibil (de cele mai multe ori lungimea șirului de simboluri obținut după aplicarea unei transformări este mai mare sau egală cu lungimea șirului inițial);
- lungimea șirului de simboluri obținut după aplicarea unui algoritm de compresie asupra rezultatului transformării să fie, de cele mai multe ori, mai mică sau egală cu lungimea șirului de simboluri obținut după aplicarea aceluiași algoritm de compresie asupra șirului netransformat;
- transformarea trebuie să fie reversibilă.

Transformările pe care le vom prezenta în continuare sunt transformări care, după aplicare, nu duc la pierdere de informație.

Compresia RLE

Să presupunem că o sursă de informație *S* generează un șir de *n* simboluri cu proprietatea că acesta conține subșiruri cu lungimea cuprinsă între a ($1 < a < n$) și b ($a < b \leq n$) simboluri, și toate simbolurile subșirurilor sunt identice. De exemplu, pentru șirul de 16 simboluri *aabcdeeeeffffgah*, subșirurile formate din același simbol care au lungimea cuprinsă între 1 și 8 sunt: *aa*, *eeee* și *ffff*.

Datorită acestor subșiruri, dimensiunea unui șir de simboluri generate de o sursă se poate reduce.

Metoda prin care se realizează comprimarea șirurilor de simboluri care conțin subșiruri formate din același simbol poartă numele de *Run Length Encoding* (*RLE - compresie în lungul șirului*).

În practică, valorile a și b se consideră a fi 1, respectiv multiplii de m', unde m' este cea mai mică putere a lui 2 mai mare sau egală cu numărul total de simboluri pe care le poate genera o sursă de informație *S*. În cazul șirului prezentat anterior, numărul total de simboluri este 8, deci valoarea lui m' este 8.

Compresia *RLE* constă din următorii pași:

- se inițializează un contor *i* cu valoarea 1 și se reține primul simbol generat de sursa *S*.
- șirul generat de sursa de informație se parcurge simbol cu simbol;
- la fiecare pas, dacă simbolul curent este identic cu cel reținut la pasul anterior:
  - atunci dacă *i* este mai mic decât m':
    - atunci incrementează valoarea contorului *i* și se trece la pasul următor;
    - altfel transmite la ieșire simbolul reținut împreună cu valoarea *i* și inițializează *i* cu 1.
  - altfel, dacă valoarea *i* este egală cu 1:
    - atunci transmite simbolul reținut și reține simbolul nou citit;
    - altfel transmite la ieșire simbolul reținut și valoarea *i*, reține noul simbol citit și inițializează pe *i* cu valoarea 1.

În momentul în care sursa de informație *S* nu mai transmite simboluri se mai parcurge încă o dată pasul din

iterație, cu deosebirea că nu se mai reține nimic, ci se transmit valorile reținute.

Se observă că pentru șirul *aabcdeeeeffffgah* se transmite șirul *a* 2 *b c d e* 4 *f* 4 *g a h*. Lungimea noului șir este 12, față 16 cât are șirul inițial.

Pentru a decodifica acest șir, se citește primul simbol și se reține, iar apoi se parcurge secvențial șirul de simboluri. La fiecare pas, dacă simbolul este un număr, atunci se transmite simbolul reținut de câte ori indică numărul citit și se reține un nou simbol, în caz contrar se transmite simbolul reținut și se citește un nou simbol. După parcurgerea șirului de simboluri se transmite ultimul simbol citit, dacă acesta nu reprezintă un număr.

În condițiile prezentate anterior, decodificarea șirului *a* 2 *b c d e* 4 *f* 4 *g a h* are loc astfel:

- reține *a*;
- transmite *a* de 2 ori și reține *b*;
- transmite *b* și reține *c*;
- transmite *c* și reține *d*;
- transmite *d* și reține *e*;
- transmite *e* de 4 ori și reține *f*;
- transmite *f* de 4 ori și reține *g*;
- transmite *g* și reține *a*;
- transmite *a* și reține *h*;
- transmite *h*.

În practică, deoarece atât simbolurile sursei de informație, cât și numerele sunt codificate cu ajutorul unor simboluri care aparțin aceluiași alfabet, apare problema că nu putem decide dacă o reprezentare este simbol sau număr. Pentru a elimina această ambiguitate, în momentul în care trebuie să transmitem un simbol și numărul de apariții consecutive ale acestuia, se va transmite simbolul de două ori, iar la decodificare, în momentul în care se citesc două simboluri identice, vom putea ști cu siguranță că trebuie să citim o reprezentare care specifică numărul de apariții consecutive ale simbolului respectiv.

Dacă m' este diferit de numărul total de simboluri m pe care le poate genera o sursă de informație S, atunci acest alfabet se va extinde cu $m - m'$ simboluri.

Pentru a reprezenta un număr care are valoarea maximă m', se folosește exact un simbol din alfabetul extins al sursei de informație S, și anume simbolul care are numărul de ordine m' în cadrul șirului de simboluri (în ordine lexicografică) ale alfabetului extins.

Considerăm numărul p cu proprietatea că $m' = 2^p$. Pentru a reprezenta un număr, care are valoarea maximă $2^{n \cdot p}$, se folosesc exact n simboluri ale alfabetului extins.

În practică, dacă numărul maxim de simboluri ale unui subșir se consideră a fi m', atunci în cel mai rău caz (când șirul de simboluri care trebuie comprimat este format numai din subșiruri formate din simboluri identice care au lungimea 2) dimensiunea șirului de simboluri transmis este cu 50% mai mare decât dimensiunea șirului original.

Există și alte metode de a elimina ambiguitatea, însă metoda prezentată este cea mai performantă dintre toate și cea mai des utilizată în practică. Metoda se mai poate îmbunătăți dacă în loc să transmitem lungimea l a șirului de simboluri identice găsit, transmitem reprezentarea numărului l - 2, deoarece șirul are cel puțin două simboluri. Transmiterea numărului l - 2 ne conduce la modificarea condiției "dacă i este mai mic decât m'" în "dacă i este mai mic decât m' + 2" din algoritmul de codificare. În algoritmul de decodificare, în momentul în care se întâlnesc două simboluri consecutive identice, se citește lungimea l a subșirului care ar trebui să fie transmis și se transmit l + 2 simboluri.

Ordinul de complexitate al acestui algoritm este $O(n)$, unde n este numărul simbolurilor generate de o sursă de informație S.

Algoritmul *RLE* se folosește de foarte multe ori în cadrul compresiei imaginilor, în combinație cu metodele *Huffman* sau *aritmetică*. În cazul fișierelor de tipul *BMP*, algoritmul *RLE* se folosește singur.

Algoritmii entropici de compresie nu pot comprima șiruri de simboluri a căror entropie este egală cu $\log_2 m$, unde m este numărul de simboluri ale alfabetului sursei care a generat șirul. Aplicarea algoritmului *RLE* înaintea unui algoritm entropic asupra unui șir de simboluri, duce, de cele mai multe ori, la micșorarea entropiei șirului, de unde reiese concluzia că șirul de simboluri a devenit *mai comprimabil*.

Transformarea delta

Fie m numărul de simboluri distincte care pot fi generate de o sursă de informație S. Vom considera mulțimea $M = \{0, 1, ..., m - 1\}$. Vom transforma șirul de n simboluri generat de sursa de informație S în șirul L format din numerele de ordine ale simbolurilor din cadrul alfabetului. Toate elementele șirului L sunt din mulțimea M.

Transformarea *delta* constă în obținerea șirului L' din șirul L folosind următoarea relație de recurență:

$L'_1 = L_1$;

$L'_i = (L_i - L_{i-1})$ ***modulo*** m, $1 < i \le n$, unde L_i reprezintă al i-lea număr al șirului de numere L.

După ce se obține șirul L', pentru fiecare element L'_i, $1 \le i \le n$, la ieșire se transmite simbolul care are numărul de ordine L'_i în cadrul alfabetului sursei de informație S.

Transformarea inversă constă în a transforma șirul L' în șirul L și se realizează folosind următoarea relație de recurență:

$L_1 = L'_1$,

$L_i = (L'_i + L'_{i-1})$ ***modulo*** m, $1 < i \le n$.

Se observă că, după aplicarea transformării, lungimea șirului de simboluri obținut după transformare este aceeași cu lungimea șirului de simboluri generat de sursa de informație S.

Algoritmul folosit în practică pentru transformarea *delta* este următorul:

```
ultim ← 0
m ← numărul de simboluri ale alfabetului sursei S
```

Alfabetul $A = \{a, b, c, d, e, f, g, h\}$
$m = 8$
ultim = 0
$S = aabcdeeeeffffgah$

Pas	Valoare *ultim*	Simbol citit	Cod simbol (*c*)	Ieșire
1	0	*a*	0	*a*
2	0	*a*	0	*a*
3	0	*b*	1	*b*
4	1	*c*	2	*b*
5	2	*d*	3	*b*
6	3	*e*	4	*b*
7	4	*e*	4	*a*
8	4	*e*	4	*a*
9	4	*e*	4	*a*
10	4	*f*	5	*b*
11	5	*f*	5	*a*
12	5	*f*	5	*a*
13	5	*f*	5	*a*
14	5	*g*	6	*b*
15	6	*a*	0	*c*
16	0	*h*	7	*h*

Figura 1

```
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   Transmite(E, Sim(S, (c - ultim) modulo m))
   ultim ← c
sfârșit cât timp
```

Algoritmul folosit în practică pentru inversarea transformării *delta* este următorul:

```
ultim ← 0
m ← numărul de simboluri ale alfabetului sursei S
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   ultim ← (ultim + c) modulo m
   Transmite(E, Sim(S, ultim))
sfârșit cât timp
```

Notația a **modulo** b semnifică restul împărțirii lui a la b.

Pentru a descrie acești algoritmi, pe lângă subalgoritmii prezentați în episodul anterior al compresiilor de date, am introdus subalgoritmii:

- `Ord(S, c)` - returnează ca rezultat numărul de ordine al simbolului `c` în cadrul alfabetului sursei de informație `S`;
- `Sim(S, i)` - returnează ca rezultat simbolul cu numărul de ordine `i` în cadrul alfabetului sursei de informație `S`.

În figura 1 se poate observa modul de transformare a șirului de simboluri *aabcdeeeeffffgah*, generat de sursa de informație *S*, care are alfabetul format din simbolurile {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*}. Se observă că șirul *aabbbbaaabaaabch* obținut după aplicarea transformării conține mai puține simboluri distincte.

Transformarea *delta* are următoarele proprietăți:

- în cazul în care în șirul inițial se află multe secvențe formate din simboluri identice, atunci acestea se transformă în secvențe de simboluri care au numărul de ordine 0 în cadrul alfabetului sursei de informație *S*;
- în cazul în care în șirul generat de sursa *S* se află multe secvențe formate din simboluri care au proprietatea că diferența dintre numerele de ordine a două simboluri consecutive este o constantă *c*, atunci acestea se transformă în șiruri de simboluri care au numărul de ordine (*c modulo m*) în cadrul alfabetului sursei de informație;
- ordinul de complexitate al transformării *delta* și al inversei acesteia este $O(n)$, unde *n* reprezintă numărul de simboluri al șirului generat de sursa *S*;
- entropia șirului de simboluri după aplicarea transformării se modifică și în cazul folosirii pentru șiruri de simboluri generate de *surse de informație multimedia* (sursă de informație care transmite șiruri de simboluri reprezentabile *audio* sau *video*) scade.

Transformarea MTF

Considerăm șirul de numere *L* construit din numerele de ordine ale simbolurilor, generate de sursa *S*, în cadrul alfabetului sursei. Fiecare element al șirului *L* aparține mulțimii $M = \{0, 1, ..., m - 1\}$, unde *m* reprezintă numărul de simboluri distincte care pot fi generate de sursa *S*. Șirul *L* are lungimea *n*, unde *n* este numărul de simboluri generate de sursa *S*.

Transformarea *MTF* (*Move To Front - mută în față*) constă în obținerea șirului *L'* din șirul *L* astfel:

- inițial se consideră *T* cu *m* elemente și $T_i = i, i \in M$;
- se parcurge secvențial șirul *L* și pentru fiecare element L_j al șirului, L'_j va primi ca valoare numărul T_k, unde *k* reprezintă poziția pe care se află elementul L_j în șirul *T*, și elementul T_k se va muta pe poziția 0 prin deplasarea la dreapta cu o poziție a elementelor $T_0, T_1, ..., T_{k-1}$;
- pentru fiecare element L'_j, $1 \le j \le n$, la ieșire se va transmite simbolul care are numărul de ordine L'_j în cadrul alfabetului sursei *S*.

Inversa transformării *MTF* constă în obținerea șirului *L* din șirul *L'* astfel:

- inițial se consideră *T* cu *m* elemente și $T_i = i, i \in M$;
- se parcurge secvențial șirul *L'* și pentru fiecare element L'_j al șirului, L'_j va primi ca valoare numărul T_k, $k = L_j$, și elementul T_k se va muta pe poziția 0 prin deplasarea la dreapta cu o poziție a elementelor $T_0, T_1, ..., T_{k-1}$;
- pentru fiecare element L_j, $1 \le j \le n$, la ieșire se va transmite simbolul care are numărul de ordine L_j în cadrul alfabetului sursei *S*.

În exemplul din figura 2 se poate observa modul de funcționare al transformării *MTF*.

În continuare vom prezenta în pseudocod algoritmii utilizați în practică pentru transformarea *MTF* și inversa acesteia.

$A = \{a, b, c, d, e, f, g, h\}$
$S = aabcdeeeeffffgah$
$T = \{0, 1, 2, 3, 4, 5, 6, 7\}$

Pas	Simbol citit	Cod simbol (L_i)	Poziția k	Valoare T	Ieșire
1	*a*	0	0	0, 1, 2, 3, 4, 5, 6, 7	*a*
2	*a*	0	0	0, 1, 2, 3, 4, 5, 6, 7	*a*
3	*b*	1	1	1, 0, 2, 3, 4, 5, 6, 7	*b*
4	*c*	2	2	2, 1, 0, 3, 4, 5, 6, 7	*c*
5	*d*	3	3	3, 2, 1, 0, 4, 5, 6, 7	*d*
6	*e*	4	4	4, 3, 2, 1, 0, 5, 6, 7	*e*
7	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
8	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
9	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
10	*f*	5	5	5, 4, 3, 2, 1, 0, 6, 7	*f*
11	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
12	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
13	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
14	*g*	6	6	6, 5, 4, 3, 2, 1, 0, 7	*g*
15	*a*	0	6	0, 6, 5, 4, 3, 2, 1, 7	*g*
16	*h*	7	7	7, 0, 6, 5, 4, 3, 2, 1	*h*

Figura 2

Deoarece șirul T conține m numere distincte care aparțin mulțimii M, putem considera că T este o permutare a mulțimii M. În practică nu se folosește șirul T, ci un alt șir W, care reprezintă inversa permutării T, astfel că pe poziția i, $i \in M$, din șirul W se află poziția pe care o ocupă simbolul cu numărul de ordine i în șirul T. Acest lucru diminuează numărul de calcule care trebuie efectuate pentru a determina poziția unui simbol în șirul T. Cu ajutorul șirului W determinarea poziției se efectuează în $O(1)$ pași, spre deosebire de cei $O(m)$ necesari în cazul folosirii șirului T.

Algoritmul pentru transformarea *MTF* este:

```
m ← numărul de simboluri ale alfabetului sursei S
W[i] ← i, i = 1, 2, ..., m
cât timp Generează(S) execută
  c ← Ord(S, CiteșteSim(S))
  Transmite(E, Sim(W[c]))
  j ← 0
  cât timp j < m execută
    dacă W[j] < W[c] atunci
      W[j] ← W[j] + 1
    sfârșit dacă
    j ← j + 1
  sfârșit cât timp
  W[c] ← 0
sfârșit cât timp
```

În cazul algoritmului de inversare al transformării *MTF* se folosește șirul T. Acest algoritm este:

```
m ← numărul de simboluri ale alfabetului sursei S
T[i] ← i, i = 1, 2, ..., m
cât timp Generează(S) execută
  c ← Ord(S, CiteșteSim(S))
  Transmite(E, Sim(T[c]))
  aux ← T[c]
  j ← c
  cât timp j > 0 execută
    T[j] ← T[j-1]
    j ← j - 1
  sfârșit cât timp
  T[0] ← aux
sfârșit cât timp
```

Transformarea *MTF* are următoarele proprietăți:

- la fel ca în cazul transformării *delta*, în situația în care șirul inițial conține multe secvențe formate din simboluri identice, acestea se transformă în secvențe de simboluri care au numărul de ordine 0 în cadrul alfabetului sursei de informație S;
- la fel ca în cazul transformării *delta*, șirul de simboluri obținut după aplicarea transformării *MTF* are aceeași lungime cu șirul inițial;
- în cazul în care în șirul generat de sursa S se află multe secvențe formate din simboluri care au proprietatea că diferența dintre numerele de ordine, în cadrul șirului T la un anumit pas, a două simboluri consecutive este 1, acestea se transformă în șiruri de simboluri care au numărul de ordine o constantă în cadrul alfabetului sursei de informație;
- ordinul de complexitate al transformării *MTF* și al inversei acesteia este $O(n \cdot m)$, unde n reprezintă numărul de simboluri al șirului generat de sursa S, iar m este numărul de simboluri ale alfabetului sursei; în cazul în care alfabetul sursei de informație are puține simboluri (de exemplu, caracterele codului *ASCII* sunt în număr de 256) și sursa generează foarte multe simboluri, atunci complexitatea transformării devine $O(n)$;
- entropia șirului de simboluri după aplicarea transformării se modifică și în cazul folosirii în combinație cu transformarea *BWT*, pe care o vom prezenta în continuare, scade.

Pentru transformarea *MTF* mai există și alte variante care mută un simbol găsit în șirul T la începutul șirului numai dacă acesta se află pe o poziție mai mare decât un număr dat l. De obicei, mutarea se face numai dacă poziția pe care se află simbolul este mai mare decât 2. Astfel se obține o transformare care duce la o creștere importantă a performanțelor algoritmilor de compresie.

Transformarea BWT

Cele trei transformări prezentate până acum se pot aplica *on-line* (la citirea unui simbol) asupra datelor, nefiind necesară memorie suplimentară pentru a stoca simbolurile generate de către o sursă de informație.

Pentru transformarea *BWT* (*Burrows-Wheeler Transform*) este necesară o cantitate de memorie suplimentară destul de mare.

Transformarea *BWT* constă din următoarele:

- șirul de simboluri generate de o sursă de informație *S* se împarte în subșiruri care au *k* simboluri fiecare, ultimul segment având (*k* ***modulo*** *n*) simboluri;
- pentru fiecare subșir se generează toate permutările circulare care se ordonează lexicografic și se construiește un șir *S'* de simboluri format din ultimul simbol al fiecărei permutări;
- în final, pentru fiecare subșir, se transmite șirul corespunzător obținut împreună cu numărul de ordine *p* al subșirului obținut după sortare.

Această transformare a fost descoperită de către cercetătorii *Michael Burrows* și *David Wheeler* în anul 1983 dar nu a putut fi implementată eficient până în anul 1994 datorită faptului că nu se cunoștea un algoritm performant de sortare a permutărilor circulare ale unui șir de simboluri.

În continuare vom prezenta modul de transformare a unui singur șir de simboluri fără a-l împărți în subșiruri.

Fie șirul de simboluri *S* = "gazeta_de_informatică". Am notat prin "_" simbolul spațiu pentru o mai ușoară înțelegere a transformării *BWT*.

Pentru șirul *S* permutările circulare sunt:

```
gazeta_de_informatică    nformatică gazeta_de_i
azeta_de_informatică g   formatică gazeta_de_in
zeta_de_informatică ga   ormatică gazeta_de_inf
eta_de_informatică gaz   rmatică gazeta_de_info
ta_de_informatică gaze   matică gazeta_de_infor
a_de_informatică gazet   atică gazeta_de_inform
_de_informatică gazeta   tică gazeta_de_informa
de_informatică gazeta_   ică gazeta_de_informat
e_informatică gazeta_d   că gazeta_de_informati
_informatică gazeta_de   ă gazeta_de_informatic
informatică gazeta_de_
```

Ordonăm lexicografic aceste permutări:

```
_de_informatică gazeta   gazeta_de_informatică
_informatică gazeta_de   icăgazeta_de_informat
a_de_informatică gazet   informatică gazeta_de_
atică gazeta_de_inform   maticăgazeta_de_infor
azeta_de_informatică g   nformatică gazeta_de_i
ă gazeta_de_informatic   ormatică gazeta_de_inf
că gazeta_de_informati   rmatică gazeta_de_info
de_informatică gazeta_   ta_de_informatică gaze
e_informatică gazeta_d   tică gazeta_de_informa
eta_de_informatică gaz   zeta_de_informatică ga
formatică gazeta_de_in
```

Dacă luăm ultimul simbol al fiecărei permutări după sortare se obține șirul *S'* = "aetmgci_dznăt_rifoeaa", iar numărul de ordine al șirului *S* este 11, primul șir având numărul de ordine 0. Șirul *S'* astfel obținut se transmite împreună cu numărul *p* = 11.

În prezent se cunoaște o clasă de algoritmi pentru sortarea permutărilor circulare ale unui șir de simboluri care au complexitatea $O(n \cdot \log n)$.

Inversa transformării *BWT* are la bază observația că șirul de simboluri *S'* este o permutare a șirului *S* și prin ordonarea lexicografică a simbolurilor acestuia se obține un șir de simboluri *F* identic cu cel care se obține dacă din fiecare șir din ordinea sortată luăm primul simbol. Așadar, dându-se un simbol S'_j în șirul *F* există un simbol *c* pe poziția *j*. Știind că aceasta este a *i*-a apariție a simbolului *c* în șirul *F*, căutăm și păstrăm a *i*-a apariție a simbolului *c* în șirul *S'*. O altă observație interesantă este aceea că, în șirul *S*, simbolul *c* precede simbolul S_j.

Dacă plecăm de la poziția *p* din șirul *S'* și efectuăm de *n* ori operația descrisă anterior, se obține un șir de simboluri identic cu șirul *S* ale cărui simboluri au fost puse în ordine inversă.

Algoritmul de obținere a șirului *S* din șirul *S'* și numărul *p* este următorul:

```
m ← numărul de simboluri ale alfabetului sursei
S' ← șirul de simboluri care trebuie inversat BWT
p ← poziția pe care o ocupă șirul S în secvența sortată
S ← șirul de simboluri care trebuie obținut
n ← lungimea șirurilor S' și S
pos[i] ← 0, i = 0, 1, ..., m - 1
i ← 0
cât timp i < n execută
  prev[i] ← pos[Ord(A ,S'[i])]
  pos[Ord(A ,S'[i])] ← pos[Ord(A ,S'[i])] + 1
  i ← i + 1
sfârșit cât timp
k ← n
i ← m-1
cât timp i ≥ 0 execută
  k ← k - pos[i]
  pos[i] ← k
  i ← i - 1
sfârșit cât timp
k ← p
i ← n-1
cât timp i ≥ 0 execută
  S[i] ← S'[k]
  k ← prev[k] + pos[Ord(A ,S'[k])]
  i ← i - 1
sfârșit cât timp
```

În algoritmul anterior, `pos[i]` reprezintă poziția corespunzătoare primei apariții a simbolului cu numărul de ordine `i` în șirul sortat al simbolurilor lui `S'`, iar `prev[k]` numărul de apariții ale simbolului cu numărul de ordine `k`. Am considerat că `A` este sursa de informație.

Algoritmul pentru inversarea transformării *BWT* are ordinul de complexitate $O(n)$, unde *n* este numărul total de simboluri generate de o sursă de informație.

Această transformare este importantă pentru că, dacă în șirul de simboluri generate de o sursă de informație sunt grupuri care încep cu același simbol, în șirul *S'* vor exista subșiruri formate din același simbol.

Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

TEOREME de programare

Clara Ionescu

Dacă nu v-ați convins până acum că modelele de rezolvare ale anumitor subprobleme constituie adevărate teoreme, pentru care atât corectitudinea cât și optimalitatea se pot demonstra, vă propunem să urmăriți următoarele tipuri de algoritmi și să analizați felul în care ați procedat până acum.

Intersecția

În numărul precedent am rezolvat câteva probleme în care *se dădea un șir* și *se cerea crearea mai multora* pe baza unei proprietăți care permitea descompunerea șirului dat în mai multe subșiruri. Evident, se poate pune întrebarea, cum procedăm când *se dau mai multe șiruri* și trebuie să creăm *un singur șir* care va conține ***intersecția*** lor. Aici prin intersecție înțelegem șirul care conține toate elementele comune, existente în toate șirurile date. Deci este o problemă care cere *selecționarea* anumitor elemente pe baza proprietății de a fi prezente în fiecare șir dat.

Evident, în diverse limbaje de programare, această problemă se rezolvă în funcție de instrumentele pe care acestea le pun la dispoziția programatorilor. De exemplu, în *Pascal* există tipul **set**, care poate simplifica mult efectuarea acestei operații, dacă tipul de bază și domeniul valorilor elementelor permit declararea șirurilor ca fiind de acest tip. Acum, ne-am propus să tratăm problema enunțată în condiții generale, independent de limbajul de programare ales, doar din punct de vedere algoritmic. De asemenea, presupunem că fiecare șir conține elemente distincte și că aceste șiruri nu sunt ordonate. Veți spune că acest lucru nu trebuie să constituie un impediment, deoarece am putea să le ordonăm și să proiectăm un algoritm simplu care determină intersecția șirurilor. Dar ordonarea, în acest caz, ar însemna o risipă de timp pe care nu întotdeauna ne-o putem permite. În concluzie, vom aplica teorema numită *selecționare* care conține în interior o *decizie*.

P24. Se consideră toți divizorii a două numere naturale. Să se determine toți divizorii comuni!

P25. Să ne amintim de o problemă rezolvată la începutul acestui serial: Cercetând lumea păsărilor din Delta Dunării, *k* specialiști au notat, fiecare, tipurile de păsări observate. Acum îi "trimitem" acolo pe ornitologii noștri atât vara, cât și iarna și le cerem să ne spună, pe baza observării speciilor prezente vara, respectiv iarna, care sunt speciile care nu migrează.

P26. Cunoscând serile libere a patru persoane, să se stabilească serile în care ar fi posibil să meargă împreună la cinema!

Prezentăm un subalgoritm general care stabilește șirul acelor elemente din șirul *X* care se găsesc și în șirul *Y*.

```
n, m: întreg                {numărul elementelor șirurilor date}
X, Y: tablou(1..n): tip element   { elementele șirurilor date}
buc: întreg                 { numărul elementelor șirului intersecție}
Z: tablou(1..n): întreg  { elementele comune în șirurile date}

subalgoritm Intersecție(n,X,m,Y,bucZ,Z):
  buc ← 0
  pentru i ← 1, n execută:
 j ← 1
    cât timp (j<=m) și (X[i]<>Y[j]) execută:
      j ← j + 1
    sfârșit cât timp
    dacă j <= m atunci
      buc ← buc + 1
      Z[buc] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Implementarea algoritmului prezentat nu ridică nici o problemă care să îl împiedice pe cititor să rezolve cu ajutorul lui problemele enunțate, deci nu le vom detalia.

În continuare vom exemplifica modul în care acest model se poate aplica în rezolvarea unor probleme care la prima vedere nu au nimic comun cu problema intersecției.

1. Vom putea aplica algoritmul într-o *decizie* unde proprietatea care trebuie verificată se definește astfel: *fiind date două mulțimi, să se stabilească dacă au sau nu cel puțin un element comun!*

2. Dacă se cere stabilirea *unui singur element comun*, avem de realizat o *selecție*.
3. Dacă se cere *determinarea unui element comun dacă există un astfel de element*, avem o *căutare secvențială*.
4. În schimb, dacă trebuie *să numărăm câte elemente comune există în două șiruri date*, avem o *numărare*.

Vom exemplifica modul de aplicare a algoritmului Intersecție pentru a răspunde la întrebarea 3. În rezolvare căutăm în șirul *X* un element care există și în *Y*. Deci vom avea o căutare secvențială care conține o decizie. Evident, algoritmul are calitatea că nu continuă căutarea după ce s-a găsit primul element comun.

```
subalgoritm Element(n,X,m,Y,există,e):
  i ← 1
  există ← fals
  cât timp (i<=n) și not există execută:
 j ← 1
    cât timp (j<=m) și (X[i]<>Y[j]) execută:
      j ← j + 1
    sfârșit cât timp
    dacă j <= m atunci
      există ← adevărat
      e ← X[i]
    altfel i ← i + 1
    sfârșit dacă
  sfârșit cât timp
sfârșit subalgoritm
```

Reuniunea

Dacă vrem să rezolvăm probleme care prelucrează mulțimi implementate cu ajutorul șirurilor, trebuie să tratăm și problema *reuniunii*.

P24. Cunoscând divizorii primi a două numere, să se determine divizorii primi ai celui mai mic multiplu comun a celor două numere!

P25. Se consideră orarul a doi profesori dintr-o școală. Să se determine orele în care cel puțin unul dintre ei poate să suplinească un eventual profesor care lipsește!

Vom prezenta și de data aceasta subalgoritmul general în care avem următoarele date:

n, m: întreg {*numărul elementelor șirurilor date*}
X, Y: tablou(1..n): tip element { *elementele șirurilor date*}
buc: întreg { *numărul elementelor șirului reuniune*}
Z: tablou(1..n): întreg { *elementele șirului reuniune*}

În rezolvare, mai întâi vom copia toate elementele din șirul *X* în șirul *Z*, inițializând lungimea *buc* cu lungimea acestui șir. Apoi, vom selecționa din șirul *Y* acele elemente care nu se află în *X*. În consecință, algoritmul conține o *copiere* și o *selecționare* care conține o *decizie*.

```
subalgoritm Reuniune(n,X,m,Y,bucZ,Z):
  Z ← X
  buc ← n
  pentru j ← 1, m execută:
 i ← 1
    cât timp (i<=n) și (X[i]<>Y[j]) execută:
      i ← i + 1
    sfârșit cât timp
    dacă i > n atunci
      buc ← buc + 1
      Z[buc] ← Y[j]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

În locul prezentării rezolvărilor problemelor, vom vedea modul în care acest subalgoritm se poate aplica în rezolvarea unor probleme clasice. De exemplu, este posibil ca într-o problemă să fie specificată cerința de *a transforma un șir de numere reale într-o mulțime*. Această cerință "tradusă" în termenii modelelor noastre poate fi formulată astfel: *să se copieze elementele unui șir într-un șir nou astfel încât acesta să conțină numai elemente distincte*. În concluzie, în șirul rezultat vom reține acele elemente din șirul dat care încă nu apar în șirul rezultat.

```
subalgoritm CreareMulțime(n,X,bucZ,Z):
  buc ← 0
  pentru i ← 1, n execută:
 j ← 1
    cât timp (j<=buc) și (X[i]<>Z[j]) execută:
      j ← j + 1
    sfârșit cât timp
    dacă j > buc atunci
      buc ← buc + 1
      Z[buc] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Concluzii

Să observăm că pentru rezolvarea unor probleme care necesită subalgoritmi pentru rezolvarea problemelor enunțate, majoritatea începătorilor ar implementa, și ei, un algoritm pătratic, dar cu două structuri repetitive de tipul **pentru**. Veți spune că amândouă abordările conduc la algoritmi pătratici... și veți avea dreptate. Totuși, există o diferență mare între o structură **pentru** și una **cât timp**. Știm că nucleul din structura **pentru** se execută de *n* ori, iar corpul structurii **cât timp** doar în cazul cel mai defavorabil are acest număr de executări. Deci, este recomandabil să nu fim neglijenți în alegerea tipului structurilor repetitive.

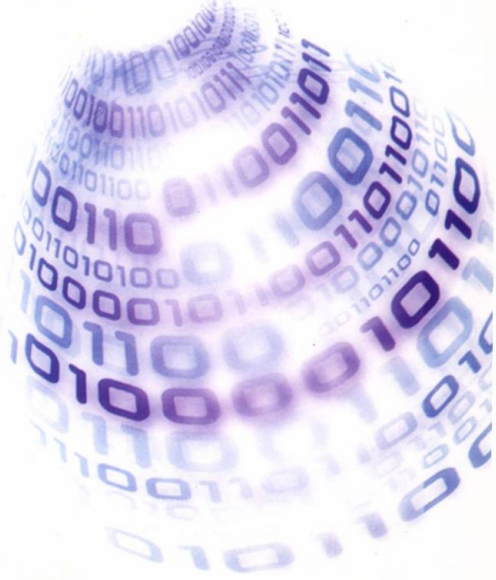
Problema intersecției și reuniunii se prezintă altfel în cazul în care șirurile sunt odornate. Dar acest subiect îl vom trata în numărul viitor.

15 feb.-30 mai 2003

CUPA SIVECO

concurs național
de software educațional

Sub patronajul Ministerului Educației și Cercetării
Partener media PC Magazine România



SIVECO ROMANIA SA

Ministerul Educației și Cercetării

PC MAGAZINE ROMÂNIA

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

SPONSOR DE AUR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

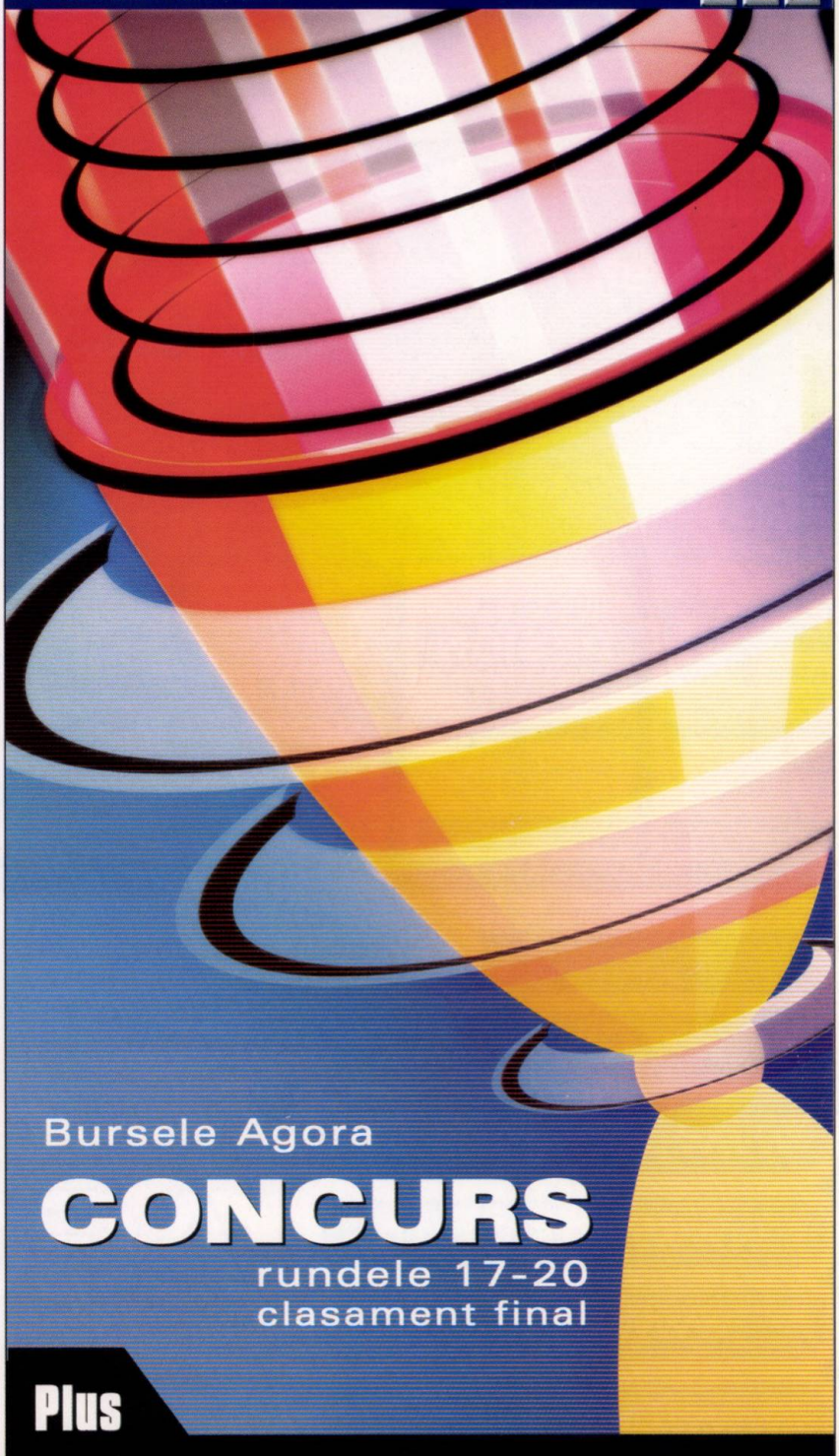
Mai 2003
Vol. 13/5

Lei 30.000

gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:



Bursele Agora

CONCURS

rundele 17-20
clasament final

Plus

probleme

olimpiada națională de informatică

bac

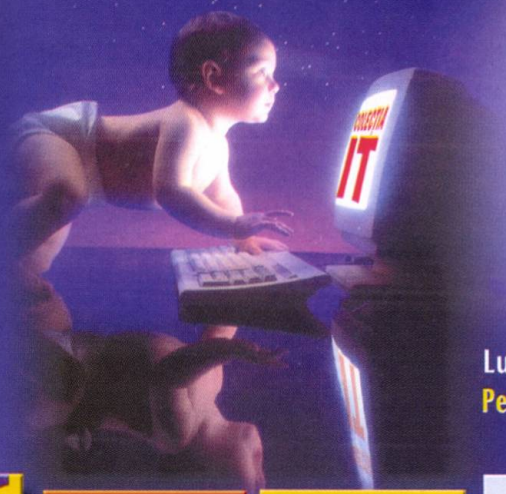
bac info 2003-răspunsurile corecte

serial

compresia datelor-ultimul episod

Editura CORINT

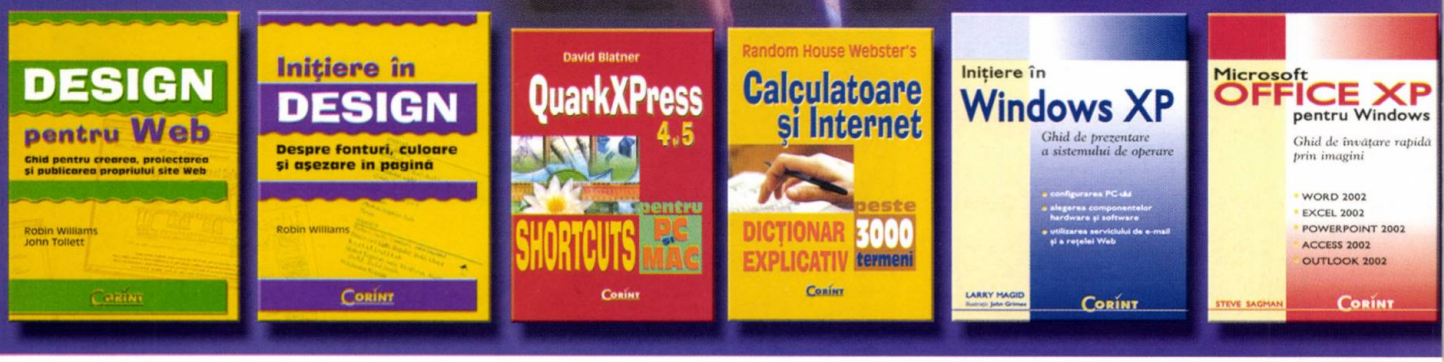
El a aflat...
Voi nu vreţi?



COLECŢIA
IT

CORINT *Difuzare:* Str. Teodosie Rudeanu nr. 21,
Tel: 222 19 49; 223 19 28; Fax: 222 40 34;
E-mail: corint@dnt.ro

Lucrări sub licenţa prestigioaselor edituri:
Peachpit Press – Pearson Technology Group,
Random House Webster's



www.agora.ro
www.pcmagazine.ro
www.netreport.ro
www.eweek.ro
www.pcconcrete.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Vol. 13, Nr. 5 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Redactor-șef:
Mihai Scorțaru
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Claudiu Soroiu

Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
400091 Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/192.422, 0265/166.516
Fax: 0265/166.290
http://www.ginfo.ro
ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 0265/162.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*", "*NetReport*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
540014 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
Florin Oprescu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

probleme.. PAG. 5

Olimpiada NAȚIONALĂ de INFORMATICĂ

~ Focșani, aprilie 2003 ~
~ 36 de probleme propuse ~

bac.. PAG. 38

BAC INFO' 2003

~ răspunsurile corecte ~
~ pentru testul grilă ~
~ din GInfo 13/4 ~

concurs.. PAG. 25

Despre desfășurarea rundelor #17-#20

Clasamentul final

Bilanțul ediției 2002/2003

serial.. PAG. 45

COMPRESIA datelor

Claudiu SOROIU

~ ultimul episod ~

~ compresia sunetelor și ~
~ a imaginilor ~

internet.. PAG. 30

Pagini WEB cu PHP 4

Mihai SCORȚARU
Claudiu SOROIU

begin.. PAG. 47

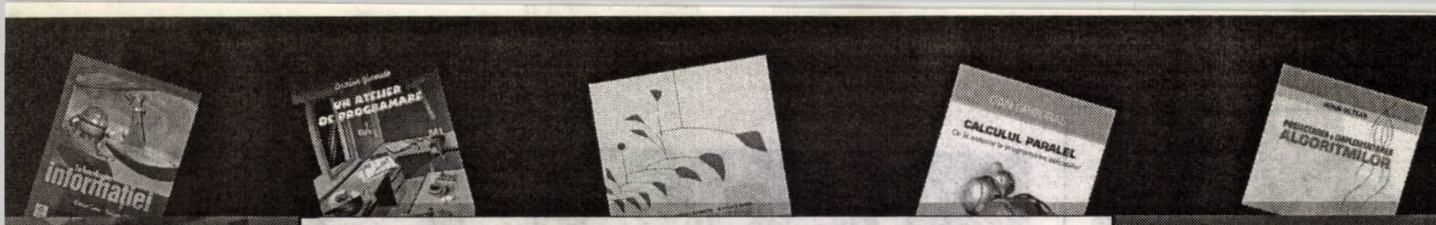
TEOREME de programare

Clara IONESCU

```
import java.io.*;

public class VacantaPlacuta{
 public static void main(String args[]){
  System.out.println("Redacția GInfo vă urează");
  System.out.println("  Vacanță plăcută!");
 }
}

import java.io.*;

public class NevedemLaToamna{
 public static void main(String args[]){
  System.out.println("Următorul număr al revistei");
  System.out.println("  va apărea în luna");
  System.out.println("   octombrie.");
 }
}
```

Vacanță plăcută!

Ne "revedem" la toamnă!



COMPUTER LIBRIS AGORA

DAN GRIGORAȘ

CALCULUL PARALEL

De la sisteme la programarea aplicațiilor

Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

PROFESOR vs. PROGRAMATOR

S-a vorbit de multe ori, exagerând în ambele sensuri, despre calitatea învățământului românesc în general, și a celui de informatică, în particular. S-au propus tot felul de soluții, care mai de care mai fanteziste dar, bineînțeles, în esență, nu s-a întâmplat nimic. De multe ori, nici măcar nu s-a încercat punerea în practică a unor astfel de soluții (cu câteva excepții, cum ar fi *cornul și laptele*).

Este evident că, până la urmă, totul se reduce la bani. Așa pare că, mai întâi, trebuie să ai "alte" resurse financiare ca să poți să-ți permiți să lucrezi în învățământ. Sau trebuie să fii un "pasionat", să fii atras sufletește de viața din școală, să-ți placă să-i inițiezi pe alții. Dar cazurile în care un tânăr să vadă mai multă fantezie în meseria de profesor decât într-o viață fără griji financiare sunt extrem de izolate... Dacă un astfel de tânăr își "permite" să devină profesor, atunci elevii se pot considera norocoși, iar dacă nu... nu vor avea posibilitatea să facă vreo comparație, să regrete sau să se bucure.

Unii absolvenți chiar încearcă să rămână, cu orice preț, în învățământ. Cîțiva reușesc să se descurce, dar marea majoritate pleacă. Unde? Informaticienii devin programatori, tinerii pleacă "prin țări străine" (eventual, la cules de căpșuni), iar cei în vârstă... rămân.

Să revenim la tinerii absolvenți care ar trebui sau ar putea să devină profesori de informatică. Cei care se pricep au de ales între a deveni programatori și a deveni profesori. Și cum informaticienii se mai pricep și la matematică, nu le este foarte greu să aleagă (din nou, nu mă refer la excepții).

Salariul unui programator începător este, în marea majoritate a cazurilor, de cel puțin cinci milioane de lei - de obicei, este mai mare. Perioada de probă durează foarte puțin (cel mult șase luni), după care salariul se mărește. După câteva luni în care și-au demonstrat calitățile, majoritatea tinerilor absolvenți ajung să aibă salarii formate din opt cifre (de fapt, chiar și începătorii primesc peste zece milioane în multe cazuri). Unii programatori au și salarii formate din doar patru cifre, dar numărul respectiv reprezintă dolari sau euro. ☺

Aceasta este o față a monedei; pe cealaltă scrie *profesor*. Să începem cu avantajele: număr redus de ore și vacanțele... Nu trebuie desconsiderat nici farmecul unor activități desfășurate cu oameni tineri, cu copii care cu vraja din ochii lor îi fac pe profesori să uite de viața din afara sălii de clasă. Dintre dezavantaje, nu amintesc decât pe cel mai important: cel mult trei milioane de lei pentru un începător și mult sub zece milioane pentru un profesor care se apropie de pensionare.

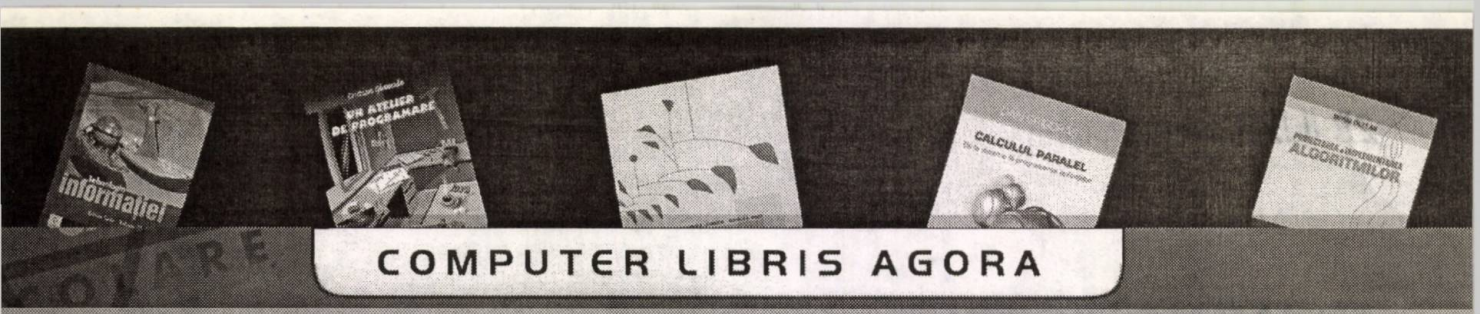
Mai există o variantă, dar foarte puțini o aleg (dacă li se permite): tinerii pot deveni asistenți universitari. Aici salariile sunt puțin mai mari, iar oportunitățile apar mai des. Cei cu adevărat bine pregătiți, vor ajunge să se afirme în afara țării și vor câștiga, în unele cazuri, mai mult decât programatorii.

Așadar, majoritatea tinerilor care se gândesc la latura materială a lucrurilor, devin programatori. Cei care doresc neapărat să rămână în învățământ se îndreaptă spre cel superior. În concluzie, așa pare că se duc în liceu cei care "nu sunt în stare" să devină programatori (vorba proverbului: cine știe - face, cine nu - învață pe alții) sau, foarte rar, cei care au resursele financiare asigurate din altă parte. Și cum nici ei nu pot supraviețui din salariul de profesor, trebuie să reușească cumva să își asigure alte venituri. Cea mai simplă metodă o reprezintă meditațiile. Și apare un cerc vicios: datorită faptului că în majoritatea timpului, profesorul este ocupat cu aceste meditații, nu prea mai are timp și de elevii de la școală sau să mai citească despre noutățile apărute în domeniul în care lucrează. Își păstrează postul pentru "renumele" de profesor (pentru ca să vină elevii la meditații).

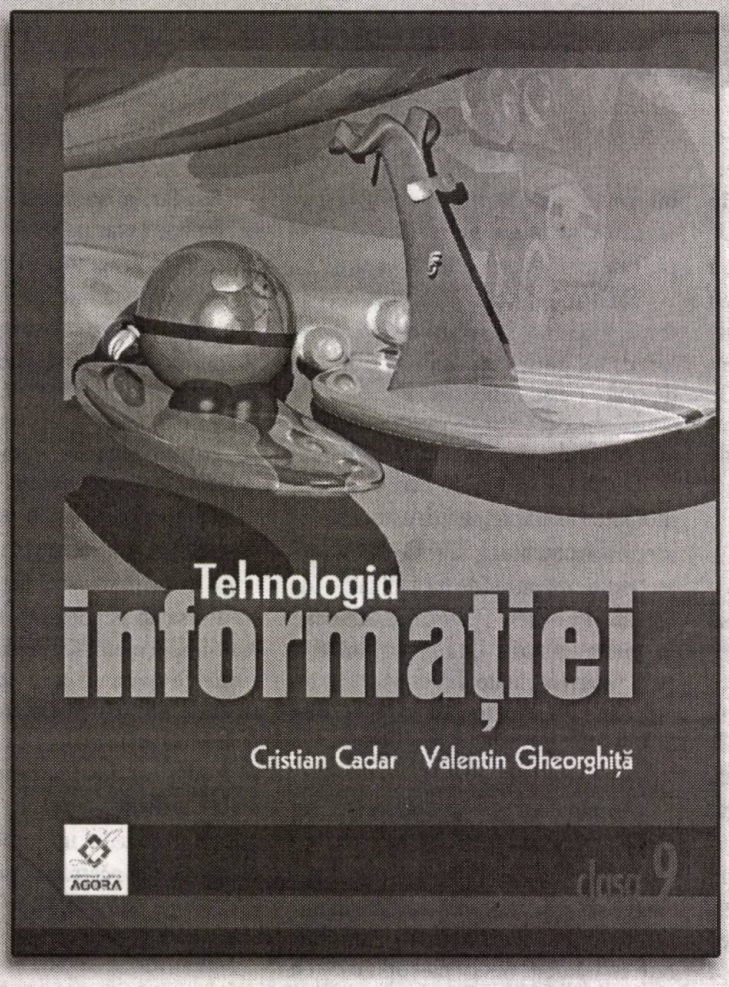
Cam atât despre profesorii noștri. Realitatea este tristă și nu stă în puterea noastră să o schimbăm. Evident, există și excepții dar... prea puține.

Închei prin a vă dori o vacanță plăcută și distracție plăcută. Vă veți reîntâlni cu *GInfo* la toamnă, în octombrie. Pe lângă noutățile la care ne vom gândi în timpul verii, vor fi câteva modificări și în caseta redacției. Parcă totuși este mai tentant să fii programator chiar și decât redactor-șef la *GInfo*, nu credeți? ☺

Mihai Scorțaru



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel.0264 - 192422

COMPUTER LIBRIS AGORA

Focșani, 18-24 aprilie 2003

Olimpiada NAȚIONALĂ de informatică 2003

Vă prezentăm în continuare enunțurile celor 36 de probleme propuse spre rezolvare la ediția din acest an a Olimpiadei Naționale de Informatică. Articolul cuprinde problemele pentru gimnaziu, cele pentru liceu, precum și cele pentru selectarea membrilor lotului național.

Clasa a V-a

P050301: Săritura cangurului

A fost odată ca niciodată un cangur care creștea, ca *Făt-Frumos* din poveste, într-un an precum alții în zece. Într-o zi a început să facă sărituri. Și a sărit pentru început șapte metri. A doua zi a sărit, în plus față de ziua precedentă, de zece ori mai mult. În a treia zi a reușit să sară, în plus față de prima zi, de zece ori mai mult decât în ziua a doua. În a patra zi a sărit, în plus față de prima zi, de zece ori mai mult decât în ziua a treia. Și așa mai departe.

Scrieți un program care calculează câți metri a sărit cangurul, în total, în n zile.

Date de intrare
Se citește de la tastatură valoarea n.

Date de ieșire
Se va afișa pe ecran câți metri a sărit cangurul, în total, după cele n zile.

Restricție
- n este un număr natural, strict mai mic decât 12.

Exemplu
Pentru $n = 3$ se va afișa: 861 metri.

P050302: Numere prime

Se numește număr prim, un număr care este divizibil doar cu 1 și cu el însuși. Astfel în intervalul [1, 30] numerele prime vor fi: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 (în total sunt zece numere prime).

Dându-se două numere n și k să se determine $2 \cdot k$ numere prime situate în centrul listei numerelor prime din intervalul $[1, n]$, în cazul în care intervalul conține un număr par de numere prime, respectiv $2 \cdot k - 1$ numere din centrul listei de numere prime, în cazul în care numărul de numere prime din interval este impar.

Dacă numărul $2 \cdot k$ (respectiv $2 \cdot k - 1$) este mai mare decât numărul de numere prime din intervalul considerat, atunci se vor afișa toate numerele prime din interval.

Date de intrare
De la tastatură se vor citi două numere n și k:
- n reprezintă marginea superioară a intervalului din care se determină numerele prime;
- k are semnificația din enunț.

Date de ieșire
Se vor afișa pe ecran numerele cerute, separate prin spațiu.

Restricții și precizări
- $1 \le n \le 10000$;
- $1 \le k \le 30$;
- $k \le n$;
- 1 nu se consideră a fi număr prim.

Exemple

Date de intrare	*Date de ieșire*
21 2	5 7 11 13
18 2	5 7 11
18 18	2 3 5 7 11 13 17
100 7	17 19 23 29 31 37 41 43 47 53 59 61 67

P050303: Poarta Orintiei

Copa bate la poarta *Orintiei*, dar poarta este programată să nu se deschidă decât după ce se introduc, într-o casetă cu s spații ($3 \le s \le 10$), s cifre strigate de portar. Portarul a strigat: "1", *Copa* a butonat 1, în primul spațiu de la stânga la dreapta. Portarul a strigat: "0", și, în timp ce *Copa* butona 0 în spațiul al doilea, 1 a devenit 2 în spațiul anterior. Portarul a strigat: "7". *Copa* scria 7 în spațiul al treilea, iar în primul spațiu, 2 devine 3, iar în al doilea spațiu, 0 devine 1. Și tot așa, până la al s-lea spațiu, când *Copa* reușește să scrie toate cifrele și apare tot codul. Și poarta se deschide,

dar... surpriză, mai era o poartă, iar codul acesteia, N, era cel mai mic număr format din cât mai multe dintre cifrele codului anterior, astfel încât nici o cifră să nu se repete.

Disperat de atâta informatizare, *Copa*, umil cetățean al *Orintiei* vă cere sprijinul să calculați cel de-al doilea cod N.

Date de intrare
De la tastatură se introduc numărul s de spații ale casetei și cele s cifre pe care le strigă portarul.

Date de ieșire
Se vor afișa pe ecran cifrele codului N neseparate prin spații.

Precizare
În cazul în care într-un spațiu al casetei se află cifra 9, după ce *Copa* va introduce următoarea cifră, cifra 9 va deveni 0.

Exemplu

Date de intrare	*Date de ieșire*
10	102456789
1	
0	
7	
9	
7	
3	
6	
9	
4	
6	

Timp de execuție: 1 secundă/test

Clasa a VI-a

P050304: Ora de sport

Profesorul de sport al clasei a *VI*-a *B* de la o școală din *Focșani* vrea la începutul orei să așeze elevii pe terenul de sport, la raport, într-o anumită ordine. Pentru acest lucru, elevii sunt bine instruiți, astfel încât, așezând pe ultimul rând n elevi, celelalte rânduri de elevi să se creeze singure după regula că pe poziția i a unui rând se va așeza un elev, după cum urmează: dacă pe rândul din spate, pe pozițiile i și $i + 1$ stau fie numai băieți, fie numai fete, atunci se va așeza o fată, iar dacă pe aceste poziții stau elevi de sexe diferite, se va așeza un băiat.

Conform acestei reguli, pe rândul cu numărul de ordine i ($i \in \{1, 2, ..., n\}$) se vor așeza i elevi.

Numărul de elevi din clasă este $n \cdot (n + 1) / 2$.

Pentru n dat și un șir de n numere 0 și 1 (0 reprezintă codificarea pentru o fată, iar 1 pentru un băiat), care reprezintă șirul de elevi de pe ultimul rând, se cere să se determine numărul de băieți din clasă.

Date de intrare
De la tastatură se citesc datele de pe două linii:

- de pe prima linie se citește numărul n;
- de pe linia a doua se citește un șir de n numere 0 sau 1, separate printr-un spațiu care reprezintă șirul de elevi de pe ultimul rând.

Date de ieșire
Pe ecran se va afișa numărul de băieți din clasă.

Restricție
- $1 \le n \le 20$.

Exemplu

Date de intrare	*Date de ieșire*
5 1 0 0 1 1	8

Explicație
Pentru exemplu avem așezarea:

```
1   0   0   1   1   ultimul rând
  1   0   1   0     rândul 4
    1   1   1       rândul 3
      0   0         rândul 2
        0           rândul 1
```

Timp de execuție: 1 secundă/test

P050305: Număr

Gigel are de rezolvat următoarea problemă: se consideră numărul natural N format din maxim nouă cifre, distincte două câte două și în care nu există cifra 0. *Gigel* va trebui să facă bilețele pentru fiecare cifră din număr, bilețele pe care le va pune într-o căciulă, conform următorului algoritm: inițial pornește de la ultima cifră a numărului (cifra unităților) și pune în căciulă bilețelul pe care este scrisă această cifră.

Dacă aceasta este o valoare pară, începe parcurgerea numărului spre dreapta, în caz contrar spre stânga, parcurgerea făcându-se cu un număr de pași egal cu cifra respectivă.

În timpul parcurgerii unui număr spre dreapta se consideră că după ultima cifră urmează prima (cea mai semnificativă cifră a numărului), după aceasta urmează a doua, etc., iar în deplasarea spre stânga după prima cifră (cea mai semnificativă cifră a numărului) urmează ultima cifră (cifra unităților), apoi penultima, etc., iar parcurgerea începe cu cifra din număr imediat de lângă cifra scrisă pe ultimul bilețel introdus în căciulă, respectând sensul parcurgerii.

De exemplu, dacă numărul nostru este 1346, *Gigel* pornește de la cifra 6, iar biletul pe care s-a scris această cifră îl pune în căciulă. El parcurge numărul spre dreapta, efectuând șase pași; el trece de cifrele: 1, 3, 4, 6, 1 și se oprește la cifra 3. Deci, în căciulă va pune bilețelul pe care este scrisă cifra 3.

Algoritmul continuă până când se termină toate bilețelele sau când ajunge la o cifră pentru care bilețelul cu valoarea respectivă a fost introdus deja în căciulă.

În cazul în care algoritmul se încheie întrucât *Gigel* a pus toate bilețelele în căciulă, se va afișa cifra de pe ultimul bilețel introdus în căciulă, iar în cazul în care *Gigel* ajunge în timpul parcurgerii la o cifră pentru care bilețelul corespunzător a fost introdus deja în căciulă, se va afișa valoarea acestei cifre.

Date de intrare
Se va citi de la tastatură numărul natural N format din cel mult nouă cifre distincte între ele și care nu conține cifra 0.

Date de ieșire
Se va afișa pe ecran cifra la care *Gigel* a ajuns în momentul opririi algoritmului.

Restricție
- $0 < N < 1.000.000.000$.

Exemple
Pentru N = 412, *Gigel* începe cu cifra 2 (bilețelul cu cifra 2 este pus de *Gigel* în căciulă); fiind valoare pară, parcurge spre dreapta și se oprește la cifra 1, bilețelul cu această cifră fiind pus în căciulă. Cifra 1 fiind impară, continuă parcurgerea spre stânga și se oprește la cifra 4 și pune astfel și ultimul bilețel în căciulă. În acest moment nu mai există bilețele și se va afișa cifra 4.

Pentru N = 1243, *Gigel* începe cu cifra 3 (bilețelul cu cifra 3 este pus de *Gigel* în căciulă); fiind valoare impară, parcurge spre stânga și se oprește la cifra 1, bilețelul cu această cifră fiind pus în căciulă. Cifra 1 fiind impară, continuă parcurgerea spre stânga și se oprește tot la cifra 3, dar nu mai există bilețelul cu cifra 3 pentru a putea fi pus în căciulă. Se va afișa deci, cifra 3.

Timp de execuție: 1 secundă/test

P050306: "Reorganizarea" numerelor

Ministerul numerelor are de câteva zile un nou șef. Acesta a dorit să facă o serie de schimbări în ministerul pe care îl conduce și a început "reorganizarea" cu mulțimea numerelor naturale în două etape: mai întâi toate numerele naturale au fost așezate fără spațiu (sau alt separator) între ele. După această primă etapă, mulțimea numerelor naturale arăta astfel:
12345678910111213141516171819202122232425262 7...

A doua etapă a "reorganizării" a constat în formarea unor noi "grupe": o grupă de o cifră, o grupă de două cifre, o grupă de trei cifre și așa mai departe. Astfel, "grupele reorganizate" sunt: 1, 23, 456, 7891, 01112, 131415, 1617181, 92021222, 324252627...

Pentru un număr natural N dat, să se afișeze prima și ultima cifră din cea de-a N-a grupă de cifre obținută după "reorganizare", valori separate printr-un spațiu.

Date de intrare
De la tastatură se citește valoarea numărului natural N.

Date de ieșire
Pe ecran se va afișa prima și ultima cifră din cea de-a N-a grupă de cifre obținută după "reorganizare", valori separate printr-un spațiu.

Restricție
- $1 \le n \le 250$.

Exemplu
Pentru N = 8 se va afișa: 9 2 (deoarece 9 și 2 sunt prima, respectiv ultima cifră din grupa a opta care este 92021222)

Timp de execuție: 1 secundă/test

Clasa a VII-a

P050307: Portocal

Oaza *Lacotrop* din deșertul *Etpas* este înconjurată de n portocali care conțin toți fructe, dispuși sub formă de cerc și numerotați de la 1 la n, în sensul acelor de ceasornic. Maimuța *Gino* pornește de la un portocal m și numără întotdeauna, în sensul acelor de ceasornic, k portocali care conțin fructe. Culege toate fructele din portocalul de pe poziția k. Continuă numărătoarea începând cu portocalul următor celui din care a cules, dar care conține fructe. În final rămâne un singur portocal p necules, în care *Gino* își face adăpost.

Cu ce portocal m trebuie să înceapă *Gino* numărătoarea pentru ca acesta să-și facă adăpostul exact în portocalul p?

Date de intrare
Fișierul de intrare **portocal.in** conține pe o singură linie, numerele n, k și p, separate printr-un singur spațiu.

Date de ieșire
Fișierul **portocal.out** conține pe prima linie numărul natural m, reprezentând portocalul cu care *Gino* începe numărătoarea.

Restricții
- $2 \le n \le 1.000$;
- $1 \le k \le 10.000$;
- $1 \le p \le 1.000$.

Exemplu

portocal.in	portocal.out
6 8 5	3

Timp de execuție: 1 secundă/test

P050308: Debarcare

În toiul pregătirilor debarcării din *Normandia* (din al doilea Război Mondial) ofițerii de contrainformații germani au observat că prin punctele de frontieră au început să circule fel de fel de indivizi care au pe centură gravate litere și cifre. După ce au reușit să prindă câțiva dintre ei și le-au

confiscat centurile au putut constata că întotdeauna pe centură se află un număr n de litere și cifre. După îndelungi "interviuri" au aflat că pe centură sunt codificate sub formă numerică, în baza 16, liniile și modurile de atac. Pentru a decodifica mesajul centura era tăiată în $\sqrt{n}$ bucăți care erau așezate una sub alta, după care se citeau caracterele de pe fiecare coloană, de sus în jos, iar numărul format pe o coloană era transformat în baza 10. Dacă numărul rezultat ar avea cifrele în ordine strict crescătoare atunci va ataca mai întâi infanteria, dacă era strict descrescător atacă prima aviația, altfel va fi un atac combinat (mixt). Numărul de linii de atac este egal cu $\sqrt{n}$.

Scrieți un program care, citind informațiile de pe o centură, să determine numărul x al liniilor de atac și modul în care se va desfășura atacul.

Date de intrare

De pe prima linie a fișierului **mesaj.in** se citește mesajul. Literele din mesaj vor fi doar litere mari.

Date de ieșire

Pe prima linie a fișierului **mesaj.out** se va scrie numărul x, iar pe următoarele x rânduri câte unul dintre cuvintele `infanterie`, `aviatie`, `mixt` în funcție de tipul de atac.

Precizare

- n este pătrat perfect, strict mai mic ca 100. Numerele formate în baza 10 au cel puțin două cifre. Literele care pot apărea sunt A, B, C, D, E, F.

Exemplu

mesaj.in	**mesaj.out**
01C7A8BAA	3 infanterie mixt aviatie

Explicație

Dacă tăiem centura și punem bucățile una sub alta obținem:

```
01C
7A8
BAA.
```

Așadar, numerele în baza 16 vor fi: `07B`, `1AA`, `C8A` a căror valori în baza 10 vor fi : `123`, `426`, `3210`.

Timp de execuție: 1 secundă/test

P050309: Domino

Indiferent de codificarea datelor necesare pentru memorarea coordonatei, N piese de domino de diferite înălțimi sunt așezate în dreptul unor coordonate, marcate de-a lungul unei linii drepte trasată pe o suprafață plană.

Acționând asupra unei piese, aceasta poate antrena în cădere la stânga sau la dreapta, un șir de alte piese, piese care vor cădea unele peste altele. O piesă este doborâtă de o altă piesă dacă este atinsă în cădere de aceasta, chiar și numai la baza ei, fiind astfel posibil ca și o piesă mai mică să doboare o alta mai mare, dacă cea mare este atinsă de cea mică în cădere.

Determinați lungimea maximă a unui șir de piese căzute care se obține prin acționarea unei piese într-un anumit sens. Aceasta este cea mai mare lungime pe orizontală acoperită integral de piese căzute.

Date de intrare

Fișierul de intrare **domino.in** conține pe prima linie N, numărul de piese, iar pe fiecare din următoarele N linii, coordonata la care se află piesa și înălțimea acesteia, despărțite printr-un spațiu.

Date de ieșire

Fișierul de ieșire **domino.out** va conține pe prima linie un număr natural reprezentând lungimea maximă a șirului de piese căzute.

Restricții și precizări

- suprafața este suficient de mare pentru ca piesele să nu cadă în afara ei;
- numărul de piese N este un număr natural din intervalul [1, 100];
- înălțimea unei piese este un număr natural din intervalul [1, 1000].

Exemple

domino.in	**domino.out**
5 40 1 8 2 30 4 1 6 4 11	14

domino.in	**domino.out**
6 53 3 62 4 15 5 20 6 60 8 75 9	12

Explicație

Pentru primul exemplu, lungimea celui mai lung șir de piese căzute este egală cu 14 și se obține prin acționarea spre dreapta a piesei așezate la coordonata 1, care va antrena în cădere piesa de la coordonata 4, iar aceasta la rândul ei pe cea de la coordonata 8.

Pentru cel de-al doilea exemplu, lungimea celui mai lung șir de piese căzute este egală cu 12 și se obține prin acționarea spre stânga a piesei așezate la coordonata 62, care va antrena în cădere piesa de la coordonata 60, iar aceasta la rândul ei pe cea de la coordonata 53.

Timp de execuție: 1 secundă/test

Clasa a VIII-a

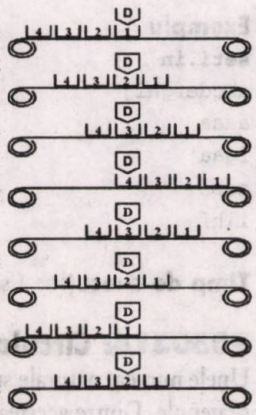
P050310: Cutii de bomboane

Pe o bandă care desfășoară o mișcare "du-te-vino" se află n cutii, inițial goale. Un dispozitiv fix suspendat deasupra benzii eliberează câte o bomboană din când în când plasând-o în cutia aflată în momentul respectiv în dreptul său.

Banda se deplasează constant astfel încât la fiecare secundă se află o altă cutie în dreptul dispozitivului D (aceasta este cutia vecină celei care s-a aflat anterior în dreptul acestuia).

Dacă $n = 4$, atunci inițial cutia având numărul 1 se află în dreptul dispozitivului, în secunda imediat următoare banda se va deplasa astfel încât cutia 2 se va afla în dreptul dispozitivului. În secundele următoare (3, 4, 5, 6, 7, 8 etc.) cutiile 3, 4, 3, 2, 1, 2 etc. se vor afla succesiv în dreptul dispozitivului.

Lungimea totală a benzii este de $2 \cdot n - 1$ ori mai mare decât lungimea unei cutii, astfel încât în timpul mișcării, va exista în permanență o cutie sub dispozitivul care eliberează bomboane.



În figura alăturată sunt reprezentate primele opt secunde de funcționare a ansamblului format din banda rulantă cu patru cutii și dispozitivul D. Se știe că, în prima secundă de funcționare a ansamblului, dispozitivul eliberează o bomboană în cutia numărul 1.

Cunoscându-se numărul n al cutiilor, durata de timp t dintre două eliberări succesive de bomboane și numărul de bomboane b eliberate în total de dispozitiv, să se determine numărul de cutii care rămân goale și numărul maxim de bomboane existente într-o cutie la sfârșitul procesului.

De exemplu, dacă $n = 6$, $t = 4$ și $b = 10$, atunci, deoarece în secundele 1, 5, 9, 13, 17, 21, 25, 29, 33, 37 dispozitivul eliberează câte o bomboană în cutiile 1, 5, 3, 3, 5, 1, 5, 3, 3 și respectiv 5, înseamnă că au rămas trei cutii goale (cutiile 2, 4 și 6) și numărul maxim de bomboane dintr-o cutie este 4.

Date de intrare

Fișierul **cutii.in** conține pe o singură linie numerele n, t și b separate prin spațiu.

Date de ieșire

Fișierul **cutii.out** va trebui să conțină pe o linie numerele c și m separate printr-un singur spațiu.

Restricții

- $1 < n < 1.000$;
- $0 < t < 1.000.000$;
- $0 < b < 1.000.000.000$.

Exemplu

cutii.in	cutii.out
4 17 5	0 2

Timp de execuție: 1 secundă/test

P050311: Matrice monoton maximală

O matrice *monotonă* este o matrice care, dacă este citită pe linii de la stânga spre dreapta sau pe coloane de sus în jos, valorile parcurse sunt crescătoare.

O submatrice este o regiune dreptunghiulară dintr-o matrice cu proprietatea că este formată din linii și coloane consecutive.

Se cere să se determine o submatrice maximală monotonă dintr-o matrice dată. Dacă există mai multe astfel de submatrice, va trebui determinată doar una dintre ele.

Date de intrare

Fișierul de intrare **matrice.in** conține pe prima linie numerele n și m, separate printr-un singur spațiu, care reprezintă numărul de linii, respectiv coloane ale matricei.

Pe fiecare a i-a linie din următoarele n linii ale fișierului se află m numere, separate printr-un singur spațiu, care reprezintă a i-a linie din matrice.

Date de ieșire

Fișierul de ieșire **matrice.out** va trebui să conțină pe prima linie două numere k și p, separate printr-un singur spațiu, care reprezintă numărul de linii și coloane ale matricei monoton maximale determinate.

Pe fiecare dintre următoarele k linii se vor afla p numere separate printr-un singur spațiu reprezentând o linie din matricea determinată.

Restricții

- $1 \le n, m \le 100$;
- elementele matricei sunt numere întregi cuprinse între 0 și 99.

Exemple

matrice.in	matrice.out
4 5	4 2
2 4 4 8 8	4 8
1 4 5 10 9	510
7 9 8 13 17	8 13
10 11 14 15 16	1415

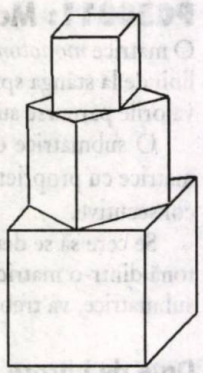
matrice.in	matrice.out
5 6	3 4
2 0 5 4 8 7	2 4 6 8
1 2 4 6 8 14	4 7 8 10
0 4 7 8 10 12	8 8 10 13
4 8 8 10 13 15	
6 6 10 12 11 16	

Timp de execuție: 2 secunde/test

P050312: Cuburi

Fie un turn format din n cuburi: primul (cel de jos) are latura L. Fiecare cub care se așează peste altul are latura mai mică decât acesta, astfel încât este poziționat cu vârfurile bazei sale exact în mijloacele laturilor bazei superioare a cubului precedent.

În figura alăturată este prezentat un turn format din trei cuburi care respectă condițiile precizate.



Dându-se numerele L, n cu semnificația descrisă mai sus și h un număr real strict pozitiv, se cere să se calculeze:

- volumul corpului obținut;
- aria totală a corpului astfel obținut (inclusiv baza inferioară a cubului de jos și cea superioară a cubului de sus);
- înălțimea turnului;
- numărul minim de cuburi care formează un turn de înălțime cel puțin h, plecând de la cubul de latură L în prima poziție (cea de jos).

Date de intrare

Fișierul de intrare **cuburi.in** conține o singură linie pe care se află numerele L, n și h separate printr-un spațiu.

Date de ieșire

Fișierul **cuburi.out** va conține pe prima linie volumul obținut, pe cea de-a doua linie aria totală a corpului, pe cea de-a treia linie înălțimea turnului și pe cea de-a patra linie numărul de cuburi necesare pentru a atinge înălțimea h sau `-1` dacă nu sunt suficiente `30.000` de cuburi.

Restricții și precizări

- L este număr real strict pozitiv;
- n este număr natural cuprins între 1 și 10.000;
- h este număr real strict pozitiv;
- calculele se vor efectua cu numere reale, iar afișarea se va face cu 5 zecimale exacte.

Exemple

cuburi.in
```
10.00 2 20
```

cuburi.out
```
1353.55339
800.00000
17.07107
3
```

cuburi.in
```
10.00 3 40
```

cuburi.out
```
1478.55339
900.00000
22.07107
-1
```

Timp de execuție: 1 secundă/test

Clasa a IX-a

P050313: Seti

Cercetătorii care lucrează la programul *SETI* au recepționat două transmisii de date foarte ciudate, date care ar putea veni din partea unor civilizații extraterestre. Primul set de date este format din zece caractere distincte, date în ordinea lor lexicografică, ce formează alfabetul extraterestru. A doua transmisie conține cuvinte din exact patru caractere.

Cercetătorii trebuie să ordoneze lexicografic cuvintele primite în a doua transmisie (conform alfabetului extraterestru).

Date de intrare

Fișierul de intrare **seti.in** conține pe prima linie cele zece caractere ale alfabetului, iar pe fiecare din următoarele linii câte un cuvânt.

Date de ieșire

Fișierul de ieșire **seti.out** va conține cuvintele ordonate, câte unul pe linie.

Restricții și precizări

- în fișierul de intrare nu sunt mai mult de 200.000 de cuvinte, iar caracterele sunt literele mici ale alfabetului englez;
- datele de intrare se presupun a fi corecte.

Exemplu

seti.in
```
abcdefghij
aaaa
fgaa
aabc
iihf
```

seti.out
```
aaaa
aabc
fgaa
iihf
```

Timp de execuție: 1 secundă/test

P050314: Circular

Unele numere naturale sunt formate doar din cifre distincte nenule. Dintre acestea, unele, numite ***numere circulare***, au următoarea proprietate: pornind de la prima cifră și numărând spre dreapta, după cifră, atâtea cifre cât indică aceasta, se determină o nouă cifră. Procedând la fel și pentru aceasta și pentru toate cele care urmează se va ajunge din nou la prima cifră. Dacă toate cifrele au fost vizitate exact o dată, numărul se numește circular. De exemplu numărul `1894256` este număr circular deoarece:

- are numai cifre distincte;
- nu conține cifra `0`;
- pornind de la `1` obținem, pe rând: `8`, `9`, `2`, `6`, `5`, `4`, `1`.

Scrieți un program care, pentru un N dat, determină câte numere circulare sunt mai mici sau egale cu N, precum și cel mai mare număr circular mai mic sau egal cu N.

Date de intrare

Pe prima linie a fișierului de intrare **circular.in** se află numărul natural N.

Date de ieșire

Fișierul de ieșire **circular.out** conține o singură linie, pe care se află numărul de numere circulare mai mici ca N,

precum și numărul circular maxim cerut, separate printr-un spațiu. Dacă nu există nici un număr circular mai mic ca N, în fișierul de ieșire se vor afișa două valori 0 separate printr-un spațiu.

Restricție

- $10 \le N < 10000000$.

Exemple

circular.in	circular.out
1894250	347 1849625

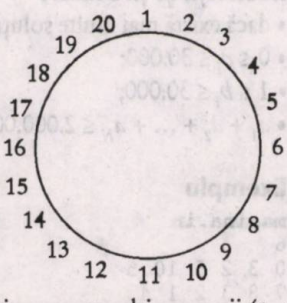
Explicație

Există 347 numere circulare mai mici decât 1894250, cel mai mare dintre acestea fiind 1849625.

Timp de execuție: 1 secundă/test

P050315: Scaune

Se consideră ns scaune numerotate de la 1 la ns, aranjate în cerc. Ca exemplu, pentru ns = 20 așezarea scaunelor este dată în figura alăturată.



Pe fiecare din aceste scaune este așezat un copil. Primul copil stă pe scaunul 1, iar ultimul pe scaunul ns.

Pe lângă cele ns scaune deja ocupate, alți nc copii ($1 \le$ nc $\le$ ns) așteaptă să se elibereze un loc.

La un moment dat un singur copil se ridică de pe scaun și pleacă. Atunci, cât timp în dreptul scaunului liber nu există un copil, toți copiii aflați în așteptare se mișcă în sens invers mișcării acelor ceasornicului, câte o poziție, până când unul din ei ocupă locul liber.

Sunt respectate următoarele condiții:

- la început toate scaunele sunt ocupate;
- fiecare copil aflat în așteptare se află inițial în dreptul unui scaun ocupat;
- când un copil avansează cu n poziții spre un loc pe scaun, toți cei care așteaptă avansează tot cu n poziții; datorită faptului că mișcarea este circulară, avansarea cu 4 poziții de la poziția 18, semnifică o deplasare în dreptul poziției 2.

Se dă o secvență a numerelor de ordine a copiilor care pleacă la fiecare moment; să se scrie un program care afișează numărul scaunului pe care s-a așezat fiecare copil care așteaptă, dacă acest lucru este posibil.

Date de intrare

Pe prima linie a fișierului de intrare **scaune.in** se află două numere, separate prin spațiu, reprezentând numărul de scaune (ns) și numărul copiilor care stau în așteptare (nc). Pe următoarele nc linii vor fi date pozițiile copiilor aflați în așteptare. În continuare, până la sfârșitul fișierului, sunt linii care descriu numerele de ordine ale copiilor care se ridică unul câte unul de pe scaune și părăsesc jocul.

Date de ieșire

Fișierul de ieșire **scaune.out** va conține nc linii, fiecare linie conținând poziția inițială de așteptare a copilului și poziția ocupată, separate printr-un spațiu. Liniile de ieșire trebuie să fie în aceeași ordine ca cele din fișierul de intrare. În cazul în care un copil nu are nici o posibilitate să se așeze, în dreptul său se va scrie 0 în fișierul de ieșire.

Restricții

- $1 \le$ ns ≤ 200;
- $1 \le$ nc $\le$ ns.

Exemplu

scaune.in	scaune.out
20 5	6 16
6	19 3
19	17 0
17	13 20
13	1 1
1	
1	
3	
20	
16	

Timp de execuție: 1 secundă/test

P050316: Criptare

Mircea și *Vasilică* vor să-și trimită mesaje pe care alții să nu le înțeleagă. Au citit ei despre spioni și modalități de a scrie mesaje și, în final, au imaginat un mod de criptare a unui mesaj care folosește *cuvânt cheie* (le-a plăcut lor denumirea asta).

Alegându-și un cuvânt cheie format numai din litere distincte, ei numără literele acestuia și împart mesajul în grupe de lungime egală cu numărul de litere ale cuvântului cheie și le așează una sub alta. Desigur, se poate întâmpla ca ultima grupă să fie incompletă, așa că o completează cu spații. Apoi numerotează literele cuvântului cheie în ordinea apariției lor în alfabetul englez. În final, rescriu mesajul astfel: coloana de sub litera numerotată cu 1, urmată de coloana de sub litera numerotată cu 2, etc. De asemenea, spațiile sunt înlocuite de caracterul '*'.

De exemplu, fie cuvântul cheie criptam și mesajul care trebuie criptat Incercam sa lucram cu coduri si criptari. Cuvântul cheie are șapte litere numerotate cu 2, 6, 3, 5, 7, 1, respectiv 4, deoarece aceasta este ordinea apriției caracterelor cuvântului în alfabetul englez. În continuare se împarte mesajul în grupuri de câte șapte litere fiecare:

Incerca|m sa lu|cram cu| coduri| si cri|ptari.

Codificăm acum mesajul:

```
2635714
Incerca
m*sa*lu
cram*cu
*coduri
*si*cri
ptari.*
```

După codificare mesajul devine:
`clcrr.Imc**pcsaoiaauuii*eamd*rn*rcstr**uci.`

Fiind date un cuvânt cheie și un mesaj criptat, decodificați mesajul trimis de *Mircea* pentru *Vasilică*.

Date de intrare
Fișierul de intrare **criptare.in** conține pe prima linie mesajul criptat, iar pe linia a doua cuvântul cheie.

Date de ieșire
Fișierul de intrare **criptare.out** conține pe prima linie mesajul decriptat.

Restricții și precizări
- lungimea mesajului este de minim 20 și maxim 1000 caractere;
- cuvântul cheie are minim 5 și maxim 20 de caractere;
- cuvântul cheie conține numai litere mici ale alfabetului.

Exemplu
criptare.in
```
clcrr.Imc**pcsaoiaauuii*eamd*rn*rcstr**uci
criptam
```

criptare.out
```
Incercam sa lucram cu coduri si criptari.
```

Timp de execuție: 1 secundă/test

P050317: Mașină
O țară are $3 \le N \le 30\,000$ orașe, numerotate de la 1 la N, dispuse pe un cerc. *PAM* tocmai și-a luat carnet de conducere și vrea să viziteze toate orașele țării. Lui *PAM* îi este frică să conducă prin locuri aglomerate, așa că ea și-a propus să meargă numai pe șoselele unde traficul este mai redus. Există șosele de legătură între oricare două orașe alăturate: între orașul 1 și orașul 2, ... , între orașul i și orașul $i + 1$, iar orașul N este legat de orașul 1. Ca să nu se rătăcească, *PAM* și-a propus să-și aleagă un oraș de început și să meargă pe șoselele respective în sens trigonometric până ajunge înapoi în orașul de unde a plecat. Dacă *PAM* pleacă din orașul K, atunci traseul ei va fi: $K, K + 1, ..., N, 1, 2, ..., K$.

Mașina lui *PAM* are un rezervor foarte mare (în care poate pune oricât de multă benzină).

În fiecare oraș, *PAM* ia toată cantitatea de benzină existentă în oraș, iar parcurgerea fiecărei șosele necesită o anumită cantitate de benzină.

Știind că *PAM* are, la începutul călătoriei, doar benzina existentă în orașul de plecare, și că, atunci când ajunge într-un oraș, ea va lua toată cantitatea de benzină disponibilă în acel oraș, să se găsească un oraș din care *PAM* își poate începe excursia, astfel încât să nu rămână fără benzină.

Se consideră că *PAM* a rămas fără benzină dacă în momentul plecării dintr-un oraș, nu are suficientă benzină pentru a parcurge șoseaua care duce la orașul următor. Dacă benzina îi ajunge "la fix" (adică la plecare are tot atâta benzină câtă îi trebuie) se consideră că *PAM* poate ajunge până în orașul următor.

Date de intrare
Fișierul de intrare **masina.in** conține pe prima linie numărul N. Pe cea de-a doua linie se găsesc N numere naturale $a_1, a_2, ..., a_N$, separate prin câte un spațiu, unde a_i reprezintă cantitatea de benzină disponibilă în orașul i. Linia a treia conține un șir de N numere naturale $b_1, b_2, ..., b_N$, separate prin câte un spațiu, unde b_i reprezintă cantitatea de benzină necesară străbaterii șoselei dintre orașele i și $i + 1$ (sau N și 1, dacă $i = N$).

Date de ieșire
Fișierul de ieșire **masina.out** va conține un singur număr s care reprezintă un oraș din care, dacă *PAM* își începe călătoria, poate completa turul țării fără a face pana prostului.

Restricții și precizări
- dacă există mai multe soluții, se va determina una singură;
- $0 \le a_i \le 30.000$;
- $1 \le b_i \le 30.000$;
- $a_1 + a_2 + ... + a_N \le 2.000.000.000$.

Exemplu
masina.in
```
6
0 3 2 5 10 5
7 8 3 2 1 4
```

masina.out
```
4
```

Timp de execuție: 0,3 secunde/test

P050318: Operații
Notăm cu c și r câtul și respectiv restul împărțirii unui număr nr la 2^k, unde k este un număr natural nenul.

Asupra numărului putem efectua succesiv una dintre următoarele operații:
- $O_1(nr, k)$ reprezintă transformarea numărului nr în numărul $2^k \cdot (2 \cdot c + 1) + r$ pentru orice rest r;
- $O_2(nr, k)$ reprezintă transformarea numărului nr în numărul $2^{k-1} \cdot c + r$ doar dacă $r < 2^{k-1}$,

Se consideră două numere naturale nenule m și n. Efectuați asupra numerelor m și n operații succesive, O_1 sau O_2, pentru valori alese ale lui k, astfel încât după un număr finit de operații cele două numere să devină egale, iar valoarea astfel obținută să fie minimă.

Date de intrare
Fișierul de intrare **operatii.in** conține pe o singură linie două numere naturale nenule m și n, separate printr-un spațiu, reprezentând cele două numere date.

Date de ieșire
Fișierul de ieșire **operatii.out** va conține:
- pe prima linie numărul natural nenul *nmin*, reprezentând valoarea minimă obținută din m și n prin aplicarea unor succesiuni de operații O_1 sau O_2;

- pe a doua linie numărul i al operațiilor efectuate asupra numărului m;
- pe fiecare dintre următoarele i linii o pereche formată din două numere reprezentând operația (1 sau 2) și respectiv valoarea lui k pentru operația respectivă, separate printr-un spațiu;
- pe linia $i + 2$ numărul j al operațiilor efectuate asupra numărului n;
- pe fiecare dintre următoarele j linii o pereche formată din două numere reprezentând operația (1 sau 2) și respectiv valoarea lui k pentru operația respectivă, separate printr-un spațiu.

Restricții

- $1 \le m, n \le 2.000.000.000$.

Exemplu

operatii.in
```
11 45
```

operatii.out
```
15
2
2 3
1 2
2
2 2
2 4
```

Timp de execuție: 1 secundă/test

Clasa a X-a

P050319: Asediu

Se aproximează o zonă de război sub forma unui cerc pe circumferința căruia se stabilesc N puncte, reprezentând comandamentele trupelor aliate, cu proprietatea că nu există trei corzi cu capetele în aceste N puncte care să fie concurente într-un punct situat în interiorul cercului. Între oricare două puncte (comandamente) există un drum sigur de acces direct. Aceste drumuri împreună cu circumferința cercului delimitează un număr de regiuni distincte. Există informații că regiunile astfel delimitate reprezintă de fapt terenuri minate de combatanții inamici. Fiecare astfel de regiune va fi cercetată amănunțit de câte un soldat aliat echipat corespunzător cu detectoare de mine.

Indicați numărul de soldați de care este nevoie pentru cercetarea tuturor regiunilor formate.

Date de intrare

Fișierul de intrare **asediu.in** conține pe prima linie numărul N al comandamentelor.

Date de ieșire

Pe prima linie a fișierului de ieșire **asediu.out** se va afla numărul T de soldați necesari pentru cercetarea tuturor regiunilor formate.

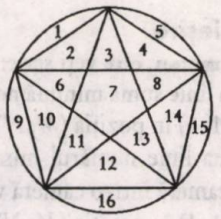
Restricție

- $2 \le N \le 2.000.000$.

Exemplu

asediu.in
```
5
```

asediu.out
```
16
```



Timp de execuție: 1 secundă/test

P050320: Muzeu

Sunteți un participant la *Olimpiada Națională de Informatică*. În programul olimpiadei intră și câteva activități de divertisment. Una dintre ele este vizitarea unui muzeu. Acesta are o structură de matrice dreptunghiulară cu M linii și N coloane.

Din orice cameră se poate ajunge în camerele vecine pe direcțiile nord, est, sud și vest (dacă aceste camere există). Pentru poziția (i, j) deplasarea spre nord presupune trecerea în poziția $(i - 1, j)$, spre est în $(i, j + 1)$, spre sud în $(i + 1, j)$ și spre vest în $(i, j - 1)$.

Acest muzeu are câteva reguli speciale. Fiecare cameră este marcată cu un număr între 0 și 10 inclusiv. Mai multe camere pot fi marcate cu același număr. Camerele marcate cu numărul 0 pot fi vizitate gratuit. Într-o cameră marcată cu numărul i ($i > 0$) se poate intra gratuit, dar nu se poate ieși din ea decât dacă arătați supraveghetorului un bilet cu numărul i.

Din fericire, orice cameră cu numărul i ($i > 0$) oferă spre vânzare un bilet cu numărul i; odată cumpărat acest bilet, el este valabil în toate camerele marcate cu numărul respectiv. Biletele pot avea prețuri diferite, dar un bilet cu numărul i va avea același preț în toate camerele în care este oferit spre vânzare.

Dumneavoastră intrați în muzeu prin colțul de nord-vest (poziția (1, 1) a matricei) și doriți să ajungeți la ieșirea situată în colțul de sud-est (poziția (M, N) a matricei). Odată ajuns acolo, primiți un bilet gratuit care vă permite să vizitați tot muzeul.

Pozițiile (1, 1) și (M, N) sunt marcate întotdeauna cu numărul 0.

Cunoscându-se structura muzeului, determinați o strategie de parcurgere a camerelor, astfel încât să ajungeți în camera (M, N) plătind cât mai puțin. Dacă există mai multe variante, alegeți una în care este parcurs un număr minim de camere (pentru a câștiga timp și pentru a avea mai mult timp pentru vizitarea integrală a muzeului).

Date de intrare

Prima linie a fișierului de intrare **muzeu.in** conține două numere întregi M și N separate printr-un spațiu, numărul de linii, respectiv de coloane, al matricei care reprezintă muzeul. Următoarele M linii conțin structura muzeului; fiecare conține N numere întregi între 0 și 10 inclusiv separate prin spații. A $(M + 2)$ - a linie conține 10 numere întregi între 0 și 10000 inclusiv, reprezentând costurile biletelor 1, 2, 3, ..., 10 în această ordine.

Date de ieşire

În fişierul **muzeu.out** veţi scrie:

- pe prima linie suma minimă necesară pentru a ajunge din poziţia (1, 1) în poziţia (M, N);
- pe a doua linie numărul minim de mutări L efectuate dintr-o cameră într-o cameră vecină, pentru a ajunge din poziţia (1, 1) în poziţia (M, N);
- pe a treia linie L caractere din multimea {N, E, S, V}, reprezentând deplasări spre nord, est, sud sau vest.

Restricţie

- $2 \le N, M \le 50$.

Exemplu

muzeu.in

```
5 6
0 0 0 0 0 2
0 1 1 1 4 3
0 1 0 0 0 0
0 1 5 1 0 0
0 0 0 1 0 0
1000 5 7 100 12 1000 1000 1000 1000 1000
```

muzeu.out

```
12
9
EEEEESSSS
```

Timp de execuţie: 1 secundă/test

P050321: Munte

Într-o zonă montană se doreşte deschiderea unui lanţ de telecabine. Staţiile de telecabine pot fi înfiinţate pe oricare dintre cele N vârfuri ale zonei montane. Vârfurile sunt date în ordine de la stânga la dreapta şi numerotate de la 1 la N, fiecare vârf i fiind precizat prin coordonata X_i pe axa Ox şi prin înălţimea H_i.

Se vor înfiinţa exact K staţii de telecabine. Staţia de telecabine i ($2 \le i \le K$) va fi conectată cu staţiile $i - 1$ şi $i + 1$; staţia 1 va fi conectată doar cu staţia 2, iar staţia K, doar cu staţia $K - 1$. Staţia 1 va fi obligatoriu amplasată în vârful 1, iar staţia K în vârful N.

Se doreşte ca lanţul de telecabine să asigure legătura între vârful 1 şi vârful N. Mai mult, se doreşte ca lungimea totală a cablurilor folosite pentru conectare să fie minimă. Lungimea cablului folosit pentru a conecta două staţii este egală cu distanţa dintre ele. În plus, un cablu care uneşte două staţii consecutive nu poate avea lungimea mai mare decât o lungime fixată L.

O restricţie suplimentară este introdusă de formele de relief. Astfel, vârfurile i şi j ($i < j$) nu pot fi conectate direct dacă există un vârf v ($i < v < j$), astfel încât segmentul de dreapta care ar uni vârfurile i şi j nu ar trece pe deasupra vârfului v. În cazul în care cele trei vârfuri sunt coliniare, se consideră toate trei ca fiind staţii, chiar dacă distanţa dintre vârfurile i şi j este mai mică decât L.

Dându-se amplasarea celor N vârfuri ale lanţului muntos, stabiliţi o modalitate de dispunere a celor K staţii de telecabine, astfel încât lungimea totală a cablurilor folosite pentru conectare să fie minimă, cu restricţiile de mai sus.

Se garantează că întotdeauna conectarea va fi posibilă.

Date de intrare

Prima linie a fişierului de intrare **munte.in** conţine trei numere întregi N, K şi L separate prin spaţii, cu semnificaţiile de mai sus. Următoarele N linii conţin coordonatele vârfurilor; linia $i + 1$ conţine coordonatele vârfului i, X_i şi H_i separate printr-un spaţiu.

Date de ieşire

În fişierul **munte.out** veţi scrie:

- pe prima linie lungimea totală minimă a cablurilor, rotunjită la cel mai apropiat numar întreg (pentru orice întreg Q, $Q + 0.5$ se rotunjeste la $Q + 1$);
- pe a doua linie K numere distincte cuprinse între 1 şi N, ordonate crescător, numerele vârfurilor în care se vor înfiinţa staţii de telecabine; dacă există mai multe variante, trebuie determinată doar una dintre ele.

Restricţii şi precizări

- $2 \le N \le 100$;
- $2 \le K \le 30$ şi $K \le N$;
- $0 \le L, X_i, H_i \le 100.000$ şi $X_i < X_{i+1}$.

Exemplu

munte.in

```
7 5 11
0 16
4 3
6 8
7 4
12 16
13 16
14 16
```

munte.out

```
22
1 3 5 6 7
```

Observaţii

Trasarea unui cablu direct între vârfurile 1 şi 5 ar fi contravenit restricţiei referitoare la lungimea maximă a unui cablu; în plus, s-ar fi obţinut o soluţie cu două staţii de telecabine în loc de trei (deci soluţia ar fi invalidă şi pentru valori mari ale lui L);

Pentru a ilustra restricţia introdusă de formele de relief, precizăm că vârfurile 1 şi 4 nu au putut fi conectate direct datorită înălţimii vârfului 3. De asemenea, vârfurile 5 şi 7 nu au putut fi conectate direct datorită înălţimii vârfului 6.

Timp de execuţie: 1 secundă/test

P050322: Partiţie

Se defineşte o partiţie a unui număr natural n ca fiind o scriere a lui n sub forma:

$$n = n_1 + n_2 + \ldots + n_k,$$

unde $k \ge 1$ şi $n_1, n_2, \ldots, n_k$ sunt numere naturale care verifică relaţia: $n_1 \ge n_2 \ge \ldots \ge n_i \ge \ldots \ge n_k \ge 1$.

Fiind dat un număr natural n, să se determine câte partiţii ale lui se pot scrie, conform cerinţelor de mai sus, ştiind că oricare număr n_i dintr-o partiţie trebuie să fie un număr impar.

Date de intrare
Fișierul **partitie.in** conține pe prima linie numărul *n*.

Date de ieșire
Fișierul **partitie.out** va conține pe prima linie numărul de partiții ale lui *n* conform cerințelor problemei.

Restricție
- $1 \le n \le 160$.

Exemplu

partitie.in	partitie.out
7	5

Explicație
Cele cinci partiții ale numărului 7 sunt:
- 1 + 1 + 1 + 1 + 1 + 1 + 1
- 1 + 1 + 1 + 1 + 3
- 1 + 1 + 5
- 1 + 3 + 3
- 7

Timp de execuție: 3 secunde/test

P050323: Rubine

Ali-Baba și cei *N* hoți ai săi au descoperit harta unei comori pe o insulă izolată. Harta este similară unui caroiaj de $L \cdot C$ regiuni (*L* linii și *C* coloane). În fiecare regiune se află un sipet fermecat, în care se găsesc rubine (0 sau mai multe), numărul de rubine din fiecare sipet fiind păstrat în elementul corespunzător al caroiajului.

Se stabilește următorul plan de acțiune:
- deplasarea hoțului se poate face dintr-o regiune într-o altă regiune vecină (o regiune are maxim opt regiuni vecine);
- fiecare hoț pleacă dintr-o anumită regiune, de coordonate date, spre regiunea în care se află *Ali-Baba* (regiunea din colțul dreapta-jos, de coordonate *L* și *C*), pe drumul cel mai scurt (lungimea unui drum fiind egală cu numărul regiunilor parcurse);
- la trecerea printr-o regiune, hoțul pune într-un sac rubinele din sipetul existent acolo;
- dacă există mai multe drumuri de lungime minimă, un hoț îl va alege pe acela pe care parcurgându-l, reușește să strângă cât mai multe rubine;
- primul hoț care se deplasează este cel cu numărul de ordine 1, urmează apoi cel cu numărul de ordine 2, 3 etc., iar în timpul deplasării unui hoț, ceilalți stau pe loc până ce acesta ajunge la *Ali-Baba*;
- sipetul fiind fermecat, în locul rubinelor luate de hoț, apar altele identice, apariția petrecându-se înaintea plecării fiecărui hoț din punctul lui de pornire spre *Ali-Baba*.

Fiecare hoț a strâns în sacul său un număr de rubine, cunoscut doar de el și ajunge cu sacul în regiunea lui *Ali-Baba*. La împărțirea rubinelor apare și *Sindbad Marinarul* care vrea o parte din saci. *Ali-Baba*, cunoscând anumite relații care există între perechi de hoți din regiune va trebui să aleagă acele relații care să implice repartizarea unui număr minim de saci lui *Sindbad*.

Se știe că alegerea unei perechi de hoți (i, j) implică repartizarea sacului hoțului *j* din "cauza" lui *i* în visteria lui *Ali-Baba*, însă din acest moment nu se mai poate folosi nici o pereche de forma (j, k), hoțul *j* fiind eliminat.

Ajutați-l pe *Ali-Baba* să aleagă ordinea perechilor astfel încât sacii care rămân să fie într-un număr cât mai mic posibil, știind că aceștia îi revin lui *Sindbad*.

Date de intrare
Fișierul de intrare **rubine.in** conține:
- pe prima linie dimensiunile caroiajului: valorile lui *L* și *C* separate printr-un spațiu;
- pe următoarele *L* linii se află câte *C* valori separate prin spațiu, reprezentând numărul de rubine din sipetul aflat în regiunea respectivă;
- pe următoarea linie se află numărul de hoți *N*;
- pe următoarele *N* linii se află coordonatele regiunilor (valori separate printr-un spațiu) în care se află inițial hoții; pe prima linie dintre acestea pentru primul hoț, pe linia a doua pentru al doilea hoț etc;
- pe următoarele linii, până la sfârșitul fișierului, se află perechi de numere care reprezintă relațiile din problemă.

Date de ieșire
Fișierul de ieșire **rubine.out** va trebui să conțină:
- pe prima linie numărul *T* de perechi folosite de *Ali-Baba*, respectându-se cerințele problemei;
- pe fiecare dintre următoarele *T* linii câte patru valori separate prin câte un spațiu: $i\ nr_1\ j\ nr_2$, unde *i* reprezintă prima valoare din pereche (hoțul cu numărul de ordine *i*), nr_1 reprezintă numărul de rubine adunate de hoțul *i*, *j* reprezintă al doilea hoț din pereche (hoțul cu numărul de ordine *j*), nr_2 reprezentând numărul de rubine strânse de hoțul *j*;
- pe următoarea linie numerele de ordine ale hoților ai căror saci au rămas la *Sindbad*, valori separate prin câte un spațiu;
- pe ultima linie se află numărul total de rubine din sacii rămași lui *Sindbad*.

Restricții și precizări
- $1 \le L, C, N \le 50$;
- fiecare element din caroiaj este un număr întreg cuprins între 0 și 10;
- întotdeauna există cel puțin o soluție;
- dacă există mai multe soluții, se va alege oricare dintre ele;
- în fiecare regiune, indiferent dacă este punctul de pornire al unui hoț sau regiunea în care se află *Ali-Baba* există câte un sipet cu un anumit număr de rubine. *Ali-Baba* nu cunoaște numărul de rubine din fiecare sac;
- nu există doi hoți care se află inițial în aceeași regiune;
- nu există succesiuni de relații circulare (exemplu: (i, j) (j, k) ... (p, t) (t, i));

• dacă există un hoț *i* care nu este în nici o relație, sacii săi vor fi repartizați direct lui *Sindbad*.

Exemplu

rubine.in
```
4 5
1 0 0 0 0
0 1 0 0 0
0 0 1 4 0
0 0 0 1 5
5
1 1
1 5
4 1
2 2
4 4
```

rubine.out
```
4
2 9 5 6
1 12 2 9
1 12 4 11
3 10 1 12
3
10
```

Timp de execuție: 1 secundă/test

P050324: Scufița

Majoritatea participanților la *ONI* 2003 au auzit în copilărie povestea *Scufiței Roșii*. Pentru cei care o știu, urmează partea a doua; pentru cei care nu o știu, nu vă faceți griji, cunoașterea ei nu este necesară pentru a rezolva această problemă. Povestea nu spune ce s-a întâmplat pe drumul pe care *Scufița Roșie* s-a întors de la bunicuță. Veți afla amănunte în continuare.

Pe drum, ea s-a întâlnit cu *Lupul* (fratele lupului care a părăsit povestea în prima parte). Acesta dorea să o mănânce, dar a decis să-i acorde o șansă de scăpare, provocând-o la un concurs de cules ciupercuțe.

Scufița Roșie se află în poziția (1, 1) a unei matrice cu *N* linii și *N* coloane, în fiecare poziție a matricei fiind amplasate ciupercuțe. *Lupul* se află în poziția (1, 1) a unei alte matrice similare. Pe parcursul unui minut, atât *Scufița*, cât și *Lupul* se deplasează într-una din pozițiile vecine (pe linie sau pe coloană) și culeg ciupercuțele din poziția respectivă. Dacă *Scufița Roșie* ajunge într-o poziție în care nu sunt ciupercuțe, va pierde jocul. Dacă la sfârșitul unui minut ea are mai puține ciupercuțe decât *Lupul*, ea va pierde jocul de asemenea. Jocul începe după ce amândoi participanții au cules ciupercuțele din pozițiile lor inițiale (nu contează cine are mai multe la începutul jocului, ci doar după un număr întreg strict pozitiv de minute de la început). Dacă *Scufița Roșie* pierde jocul, *Lupul* o va mânca.

Înainte de începerea jocului, *Scufița Roșie* l-a sunat pe *Vânător*, care i-a promis că va veni într-un sfert de oră (15 minute) pentru a o salva. Deci *Scufița Roșie* va fi liberă să plece dacă nu va pierde jocul după 15 minute.

Din acest moment, scopul ei este nu numai să rămână în viață, ci și să culeagă cât mai multe ciupercuțe, pentru a le duce acasă (după ce va veni, vânătorul nu o va mai lăsa să culeagă).

Lupul, cunoscut pentru lăcomia sa proverbială, va alege la fiecare minut mutarea în câmpul vecin cu cele mai multe ciupercuțe (matricea sa este dată astfel încât să nu existe mai multe posibilități de alegere la un moment dat).

Povestea spune că *Scufița Roșie* a plecat acasă cu coșulețul plin de ciupercuțe, folosind indicațiile date de un program scris de un concurent la *ONI*2003 (nu vom da detalii suplimentare despre alte aspecte, cum ar fi călătoria în timp, pentru a nu complica inutil enunțul problemei). Să fi fost acest program scris de dumneavoastră? Vom vedea...

Scrieți un program care să o ajute pe *Scufița Roșie* să rămână în joc și să culeagă cât mai multe ciupercuțe la sfârșitul celor 15 minute!

Date de intrare

Fișierul **scufita.in** are următoarea structură:

- pe prima linie se află numărul natural *N* care reprezintă numărul de linii și de coloane ale celor două matrice;
- pe următoarele *N* linii se află matricea din care culege *Scufița Roșie*; fiecare linie va conține *N* numere naturale, separate prin câte un singur spațiu; un astfel de număr arată câte ciupercuțe se află în poziția corespunzătoare;
- pe următoarele *N* linii se află matricea din care culege *Lupul*; fiecare linie va conține *N* numere naturale, separate prin câte un singur spațiu; un astfel de număr arată câte ciupercuțe se află în poziția corespunzătoare.

Date de ieșire

Fișierul **scufita.out** va trebui să conțină pe prima linie numărul total de ciupercuțe culese de *Scufița Roșie* iar pe următoarea linie o succesiune de caractere din mulțimea {N, E, S, V} care reprezintă direcțiile pe care s-a deplasat *Scufița Roșie* (N - *Nord*; E - *Est*; S - *Sud*; V - *Vest*). Aceste caractere nu vor fi separate prin spații.

Restricții și precizări

- $4 \le N \le 10$;
- valorile din cele două matrice sunt numere naturale mai mici decât 256;
- nici *Scufița Roșie* și nici *Lupul* nu vor părăsi matricele corespunzătoare;
- după ce unul din jucători culege ciupercuțele dintr-o poziție, în poziția respectivă rămân 0 ciupercuțe;
- *Scufița Roșie* va avea întotdeauna posibilitatea de a rezista 15 minute;
- poziția (1, 1) este situată în colțul de *Nord-Vest* al matricelor.

Exemplu

scufita.in
```
4
2 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

scufita.out
```
137
SSSEEENVVNEENVV
```

Explicație
Scufița Roșie a efectuat aceleași mutări cu cele efectuate de *Lup* și a avut tot timpul o ciupercuță în plus. În final ea a cules toate ciupercuțele din matrice.

Timp de execuție: 1 secundă/test

Clasele a XI-a și a XII-a

P050325: A007

Agentul 007 are de distrus o tabără de teroriști. Tabăra de teroriști este formată din mai multe obiective (depozite de muniție, pavilioane pentru teroriști etc.), considerate punctiforme în plan.

Agentul 007 primește de la serviciul de informații o hartă cu n obiective din tabăra teroriștilor, date prin coordonatele carteziene.

Pe lângă hartă, agentul 007 mai primește și o armă specială (construită pentru această misiune). Arma primită are două țevi și permite tragerea simultană pe aceeași direcție (rectilinie), dar în sens invers a două rachete cu aceeași viteză.

După ce se trage cu arma, odată cu atingerea unei ținte explodează și cealaltă rachetă (chiar dacă aceasta din urmă nu și-a atins ținta).

Agentul 007 vrea să distrugă tabăra cât mai repede și cu cât mai puține rachete. Pentru acest lucru el studiază posibilitatea să se așeze într-un punct din tabără (diferit de obiective) care să permită trageri eficace, adică la fiecare tragere să distrugă câte două obiective simultan.

Determinați dacă este posibil să se găsească un astfel de punct.

Date de intrare
În fișierul **a007.in** pe prima linie se află numărul de teste k, după care urmează date pentru fiecare test. Pentru fiecare test, pe o linie se află numărul n, iar pe următoarele n linii sunt coordonatele obiectivelor din tabăra teroriștilor (separate printr-un spațiu în ordinea abscisă ordonată).

Date de ieșire
În fișierul **a007.out** se vor scrie k linii, pe fiecare linie se va scrie 1, dacă există soluție și 0 dacă nu există soluție. În cazul în care există soluție se va scrie în continuare pe aceeași linie, separate printr-un spațiu, coordonatele punctului cerut (numere reale trunchiate la patru zecimale, în ordinea abscisă ordonată).

Restricții și precizări
- $0 \le n \le 10.000$;
- $1 \le k \le 3$;
- coordonatele punctelor sunt întregi din intervalul [-10.000, 10000];
- un obiectiv este distrus dacă racheta explodează exact în punctul corespunzător lui;
- coordonatele punctelor sunt distincte.

Exemplu

a007.in
```
2
4
10 0
10 10
0 10
0 0
6
0 0
10 0
2 10
12 0
5 0
7 0
```

a007.out
```
1 5.0000 5.0000
0
```

Timp de execuție: 0,2 secunde/test

P050326: Asmin

Se consideră un arbore (graf conex aciclic) cu N vârfuri, fără rădăcină fixată. Drept rădăcină, poate fi ales oricare dintre vârfuri. Să presupunem că a fost ales vârful cu numărul T. Între oricare vârf și T există un drum unic care conține fiecare vârf al arborelui cel mult o singură dată (un drum între vârfurile i și j este o secvență de vârfuri, care începe cu i, se termină cu j, iar între oricare două vârfuri consecutive există o muchie în arbore).

Fiecărui vârf i (inclusiv T) trebuie să i se asocieze o valoare V_i, mai mare sau egală cu 0, astfel încât suma valorilor vârfurilor de pe drumul dintre i și rădăcina T, împărțită la K, să dea restul R_i. Se definește costul arborelui cu rădăcina fixată în T, C_T, ca fiind suma valorilor asociate fiecărui nod. Dintre toate posibilitățile de alegere a valorilor V_i care respectă condiția precizată anterior, se va alege aceea pentru care C_T este minim.

Se constată ușor că alegând o altă rădăcină, de exemplu, vârful S (diferit de T), C_S nu este neapărat egal cu C_T.

Dându-se un arbore cu N vârfuri, un număr întreg K și valorile R_i, $i = 1, 2, ..., N$, corespunzătoare fiecărui vârf, determinați acele vârfuri T care pot fi alese drept rădăcină, pentru care costul C_T este minim (adică $C_T \le C_S$, oricare ar fi S diferit de T), precum și costul respectiv.

Date de intrare
Pe prima linie a fișierului de intrare **asmin.in** se află două valori întregi: N și K. Pe următoarele N - 1 linii se află câte două numere întregi a și b, separate printr-un spațiu, având semnificația că există muchie între vârfurile a și b. Vârfurile sunt numerotate de la 1 la N. Pe următoarea linie se află N numere întregi, reprezentând valorile R_i, $i = 1, ..., N$.

Date de ieșire
Pe prima linie a fișierului de ieșire **asmin.out** se vor afișa două valori întregi: C și M. C reprezintă costul minim posibil al arborelui. M reprezintă numărul de vârfuri care pot fi alese drept rădăcină și pentru care se obține costul C. Pe a doua linie se află M numere întregi separate prin câte un spațiu, scrise în ordine crescătoare, reprezentând numerele vârfurilor care pot fi alese ca rădăcină, astfel încât să se obțină costul C.

Restricții

- $2 \le N \le 16.000$;
- $2 \le K \le 1.000$;
- $2 \le R_i \le K - 1$.

Exemplu

asmin.in

```
5 3
1 2
1 3
2 4
2 5
0 1 2 1 0
```

asmin.out

```
5 2
1 5
```

Timp de execuție: 0,2 secunde/test

P050327: Căutare

Știm cu toții ce este un arbore binar de căutare. Este acel arbore binar în care informația din orice nod este mai mare decât informațiile nodurilor din subarborele stâng al nodului respectiv și mai mică decât cele din subarborele drept.

Când se caută o informație într-un arbore binar de căutare, se începe din rădăcina arborelui și valoarae căutată se compară cu informația din rădăcină.

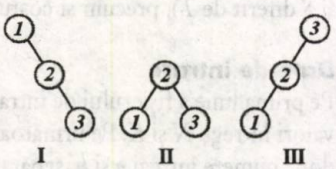
Dacă informația căutată e mai mică decât informația din rădăcină, se continuă căutarea în subarborele stâng, iar dacă e mai mare se continuă căutarea în subarborele drept.

Dacă cele două informații sunt egale, căutarea se termină cu succes.

Dacă în direcția în care continuăm căutarea (stânga sau dreapta) subarborele nu mai are noduri, înseamnă că informația căutată nu se găsește în arbore. Evident, numărul de comparații efectuate la o căutare depinde de distanța dintre rădăcină și nodul în care se găsește informația, respectiv cel la care putem decide că informația nu se află în arbore.

Ce s-ar întâmpla dacă înainte de a construi arborele binar de căutare am ști ce informații urmează să fie căutate în el? Nu cumva am putea construi arborele în așa fel încât să minimizăm numărul de comparații efectuate?

De exemplu, cu informațiile 1, 2 și 3 putem construi un arbore binar de căutare în următoarele trei moduri (din totalul de cinci posibile).



Dacă vom căuta informațiile (1, 1, 3, 1, 2), atunci timpul total de căutare este pentru arborele din stânga 1 + 1 + 3 + 1 + 2 = 8, pentru arborele din centru 2 + 2 + 2 + 2 + 1 = 9, iar pentru arborele din dreapta 3 + 3 + 1 + 3 + 2 = 12.

Deci arborele din stânga este cel adecvat pentru căutările noastre, iar timpul total de căutare minim este 8.

De remarcat că, dacă se caută o informație care nu există în arbore, atunci numărul de comparații efectuate la căutare este egal cu nivelul ultimului nod interogat. De exemplu, dacă se caută informația 4 pe cei trei arbori, atunci timpii vor fi: 3, 2, respectiv 1 (de la stânga la dreapta).

Scrieți un program care, pentru anumite informații căutate, determină timpul total de căutare minim.

Date de intrare

Din fișierul **cautare.in** se citește de pe prima linie numărul T al testelor. În fișier urmează cele T teste. Pentru fiecare test în fișier sunt scrise următoarele linii:

- prima linie conține N, numărul de noduri ale arborelui de căutare și M, numărul de interogări, separate prin spațiu;
- pe următoarea linie urmează N numere întregi distincte, separate prin câte un spațiu, reprezentând informațiile din nodurile arborelui;
- pe fiecare dintre următoarele M linii sunt câte două numere întregi separate printr-un spațiu, reprezentând un număr căutat și respectiv de câte ori a fost căutat (între 1 și 100).

Date de ieșire

În fișierul **cautare.out** se va scrie, pentru fiecare test câte o linie care conține timpul total minim de căutare pentru interogările din testul respectiv.

Restricții și precizări

- $0 < T, N < 101$;
- $0 < M < 1001$;
- informațiile nodurilor arborelui sunt numere întregi din intervalul [-10000, 10000];
- informațiile căutate sunt numere întregi cuprinse în intervalul [-1000000, 1000000] și pot fi căutate de un număr de ori cuprins între 1 și 100.

Exemplu

cautare.in

```
3
3 3
1 2 3
1 3
2 1
3 1
3 3
1 2 3
1 50
2 49
3 51
3 2
1 2 3
2 1
4 20
```

cautare.out

```
8
251
22
```

Timp de execuție:

- *Microsoft Windows*: 1,5 secunde/test
- *Linux*: 0,3 secunde/test

P050328: Inter

În țara *Smar* sunt N autostrăzi, sub forma unor drepte în plan. Se știe că la intersecții de drumuri (care includ și autostrăzi) există un risc ridicat de accidente. De aceea polițiștii din această țară au hotărât stabilirea unei zone compacte care să includă toate intersecțiile și în care să se supravegheze atent circulația. Din motive financiare zona trebuie să fie de perimetru minim.

Scrieți un program care să determine aria zonei de supraveghere alese.

Date de intrare

Fișierul de intrare **inter.in** va conține pe prima linie numărul de autostrăzi, iar de pe fiecare dintre următoarele N linii se vor citi câte patru numere reale, separate prin câte un spațiu, reprezentând coordonatele a două puncte distincte care determină câte o dreaptă. Ele sunt date în ordinea $X_1\ Y_1\ X_2\ Y_2$, adică abscisa și ordonata primului punct, apoi abscisa și ordonata celui de-al doilea punct.

Date de ieșire

Fișierul de ieșire **inter.out** va conține o singură linie pe care se va afla un număr real, cu două zecimale exacte (rotunjit), care reprezintă aria zonei alese pentru supraveghere.

Restricții și precizări

- între oricare două autostrăzi există exact o intersecție;
- aria suprafeței de supraveghere este întotdeauna strict pozitivă;
- $2 \le N \le 5001$.

Exemplu

inter.in	inter.out
4 0 0 1 0 0 0 0 2 0 2 1 0 -2 0 0 1	3.00

Timpi de execuție:

- *Microsoft Windows*: 1,5 secunde/test
- *Linux*: 0,2 secunde/test

P050329: Număr

Se consideră un număr natural x, scris în baza 10, care conține exact n cifre. Scrieți un program care să determine cel mai mic număr natural strict mai mare decât x, care are aceleași cifre ca și numărul x și care este palindrom.

Date de intrare

Fișierul de intrare **nr.in** conține două linii. Pe prima linie este scris n, numărul de cifre ale numărului x. Pe cea de a doua linie sunt scrise cele n cifre ale lui x.

Date de ieșire

Fișierul de ieșire **nr.out** conține o singură linie pe care se află cel mai mic număr natural strict mai mare decât x, care are aceleași cifre ca și numărul x și care este palindrom. Dacă nu există soluție, pe prima linie a fișierului de ieșire va fi scrisă valoarea 0.

Restricții și precizări

- $1 \le N \le 1000$;
- numim palindrom un număr care citit de la stânga la dreapta, cât și de la dreapta la stânga este același (de exemplu 1331, 12321 etc);
- prima cifră a unui număr trebuie să fie nenulă;
- prin aceleași cifre se înțelege că fiecare cifră apare în rezultat de același număr de ori ca și în numărul x.

Exemple

nr.in	nr.out
5 12022	0
5 12200	20102

Timp de execuție: 0,1 secunde/test

P050330: Proc

O aplicație care trebuie executată pe un calculator multiprocesor constă din N fragmente de cod independente, care pot fi rulate în paralel. Fiecare fragment trebuie executat în totalitate pe un singur procesor. Din dorința de a paraleliza cât mai mult aplicația, fragmentele de cod au dimensiuni mici și, în consecință, timpi de execuție mici. Mai precis, execuția fiecărui fragment durează ***unul*** sau ***două*** cicluri de ceas pe un procesor de tipul *Pentium* IV.

Sistemul pe care urmează să fie executată aplicația constă din P procesoare. Dar, spre deosebire de majoritatea sistemelor de acest fel, cele P procesoare au viteze de execuție diferite. Primul procesor este un *Pentium* IV și este cel mai rapid. Al doilea procesor este de două ori mai lent decât primul, al treilea de trei ori mai lent, ..., al i-lea procesor este de i ori mai încet decât primul. În aceste condiții, timpul de execuție al fiecărui fragment de cod diferă, în funcție de procesorul pe care va fi executat. Să presupunem că un segment de cod are timpul de execuție T (unde T este 1 sau 2) pe primul procesor. Atunci timpul de execuție al procesorului cu numărul de ordine i va fi $i \cdot T$.

Știind că fragmentele de cod pot fi executate în orice ordine și pe orice procesor și că orice procesor poate executa, la un moment dat, un singur fragment de cod, determinați timpul minim după care se va termina execuția aplicației (adică a tuturor fragmentelor de cod). Timpul după care se termină aplicația este egal cu maximul dintre timpii după care fiecare procesor redevine disponibil. Timpul după care un procesor redevine disponibil este egal cu suma timpilor de execuție a fragmentelor de cod rulate pe procesorul respectiv.

Date de intrare

Fișierul de intrare **proc.in** conține pe o singură linie trei numere întregi, separate printr-un singur spațiu: N - numărul de fragmente de cod, K - numărul de fragmente de cod care au timpul de execuție pe un *Pentium* IV, egal cu 1 (implicit, $N - K$ fragmente de cod au timpul de execuție egal cu 2) și P - numărul de procesoare ale sistemului.

Date de ieșire

Fișierul **proc.out** va conține o singură linie pe care se află timpul minim după care se termină de executat aplicația.

Restricții

- $0 \le K \le N \le 1.000.000.000$;
- $0 \le K \le N \le 65535$.

Exemplu

proc.in	proc.out
4 3 2	4

Explicație

Pe primul procesor se execută un fragment de cod cu timpul de execuție (calculat pe un *Pentium* IV) egal cu 1 și un fragment de cod cu timpul de execuție egal cu 2 ⇒ timpul după care acest procesor devine disponibil este 1 · 1 + 1 · 2 = 3. Pe al doilea procesor se execută două fragmente de cod cu timpul de execuție (calculat pe un *Pentium* IV) egal cu 1 ⇒ timpul după care acest procesor devine disponibil este 2 [*numărul de fragmente*] · (2 · 1) [*timpul de execuție al fiecărui fragment pe procesorul* 2] = 4.

Timpi de execuție:

- *Microsoft Windows*: 1,5 secunde/test
- *Linux*: 0,2 secunde/test

Baraj

P050331: Excursie

În una dintre zile, la *Olimpiada de Informatică* se organizează P excursii atractive. La aceste excursii participă în total N persoane. Pentru simplitate, persoanele au fost numerotate de la 1 la N, primele K persoane fiind ghizii. O persoană se poate înscrie la exact una dintre cele P excursii organizate.

Pentru a evita surprizele neplăcute (insuficiente mijloace de transport, insuficiente locuri la restaurant etc) organizatorii intenționează să studieze toate configurațiile care pot să apară în urma înscrierilor participanților, considerând că în fiecare excursie va exista cel puțin un participant.

Scrieți un program care să determine numărul de configurații distincte care se pot obține după înscrierea celor N persoane la cele P excursii organizate, astfel încât cei K ghizi să fie înscriși în excursii diferite.

Date de intrare

Fișierul de intrare se numește **ex.in** și conține o singură linie pe care se află trei numere naturale separate prin câte un spațiu: N K P (reprezentând numărul de persoane, numărul de ghizi și respectiv numărul de excursii).

Date de ieșire

Fișierul de ieșire **ex.out** conține o singură linie pe care se află numărul de configurații distincte.

Restricții

- $0 \le K \le P \le N \le 1.000.000.000$;
- într-o configurație nu contează ordinea excursiilor sau ordinea în care se înscriu persoanele la o excursie.

Exemplu

ex.in	ex.out
5 3 4	7

Explicație

Cele șapte configurații distincte sunt:

```
(1,4)(2)(3)(5)
(1)(2,4)(3)(5)
(1)(2)(3,4)(5)
(1)(2)(3)(4,5)
(1,5)(2)(3)(4)
(1)(2,5)(3)(4)
(1)(2)(3,5)(4)
```

Timpi de execuție:

- *Microsoft Windows*: 0,5 secunde/test
- *Linux*: 0,1 secunde/test

P050332: Robot

Lucrați la o firmă care produce microprocesoare. Pentru asamblarea microprocesoarelor, firma utilizează un robot constituit dintr-un singur braț.

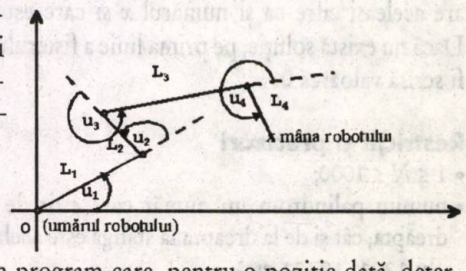
Brațul este fixat la unul dintre capete ("umărul") într-un punct plasat în centrul platformei de lucru, iar la celălalt capăt are un dispozitiv de lungime neglijabilă cu care poate "culege" componentele de pe platforma de lucru ("mâna"). Brațul se poate mișca numai în plan orizontal, deasupra platformei de lucru.

Brațul este constituit dintr-o succesiune de N segmente rigide de lungimi L_1, L_2, ..., L_N, conectate prin puncte de articulație. Mai exact, primul segment este conectat printr-un punct de articulație în umărul robotului, al doilea segment este conectat printr-un punct de articulație de primul segment etc. Ultimul segment este conectat printr-un punct de articulație de penultimul segment și are la celălalt capăt "mâna". Un punct de articulație permite rotația liberă (la orice unghi) a segmentului conectat în acel punct de articulație.

Pentru asamblarea unui microprocesor, robotul trebuie să culeagă succesiv componentele acestuia de pe platforma de lucru. Fiecare componentă are o poziție bine determinată pe platforma de lucru, prin coordonatele sale relativ la un sistem de coordonate cartezian, cu centrul în "umărul robotului".

Rolul dvs. în firmă este de a programa mișcările robotului. În acest scop, pentru fiecare componentă pe care robotul o va culege trebuie să specificați "configurația" brațului robotului care să permită atingerea componentei respective (mâna robotului să fie plasată deasupra poziției în care se află componenta).

Configurația brațului robotului este dată de unghiurile dintre segmentele brațului rigid.



Scrieți un program care, pentru o poziție dată, determină o configurație pentru brațul robotului care să-i permită acestuia să culeagă componenta din poziția respectivă, dacă este posibil.

Date de intrare

Fișierul de intrare **robot.in** conține pe prima linie un număr natural N, care reprezintă numărul de segmente din care este format brațul robotului.

Pe fiecare dintre următoarele N linii se află câte un număr natural. Numărul aflat pe a $(i + 1)$ - a linie reprezintă lungimea celui de-al i-lea segment al brațului robotului.

Pe ultima linie se află două numere întregi x și y, separate prin câte un spațiu, reprezentând coordonatele poziției la care trebuie să ajungă "mâna" robotului.

Date de ieșire

Fișierul de ieșire **robot.out** va conține o singură linie pe care se află valoarea 0 dacă nu este posibil ca mâna robotului să ajungă în poziția x, y. Dacă problema are soluție, fișierul de ieșire conține N linii. Pe linia i se află valoarea reală u_i care reprezintă unghiul dintre segmentul i și segmentul $i - 1$ (pentru orice i de la 2 la N), iar valoarea u_1 reprezintă unghiul pe care segmentul 1 îl formează în umărul robotului cu axa Ox.

Restricții și precizări

- $0 \le N \le 10.000$;
- $1 \le L_i \le 200$;
- $0 \le u_i < 360$;
- $-10.000 \le x, y \le 10.000$;
- unghiurile se măsoară în sens trigonometric și sunt exprimate în grade.

Exemple

robot.in	robot.out
3 10 5 25 15 20	125.6725 0 252.5424

robot.in	robot.out
3 10 5 25 2 4	0

Timpi de execuție:

- *Microsoft Windows*: 0,3 secunde/test
- *Linux*: 0,1 secunde/test

P050333: Telegraf

Până nu demult, comunicația la distanță se făcea cu ajutorul telegrafului. Folosind telegraful, se pot transmite două tipuri de semnale: punct și linie. În general, dorim să transmitem texte formate din litere ale alfabetului latin și cifre (în total, 36 de simboluri). Trebuie deci să folosim o codificare, adică să asociem fiecăruia dintre cele 36 de simboluri o succesiune distinctă de linii și puncte. Pentru a putea decodifica o succesiune recepționată de linii și puncte, este necesar ca nici un simbol să nu aibă o codificare identică cu începutul codificării pentru un alt simbol. Să considerăm câteva exemple (presupunând că nu vrem să transmitem decât literele A, B, C):

- exemplul 1: A = ··; B = ·–; C = –.
- exemplul 2: A = ·––; B = ·–; C = –.
- exemplul 3: A = ·–··; B = –·; C = ··–·.

Primul exemplu reprezintă o codificare corectă. Al doilea exemplu reprezintă o codificare greșită, pentru că începutul codificării lui A este identic cu codificarea pentru B (deci, o secvență de genul ··– este ambiguă, putând însemna fie A fie BC). Al treilea exemplu reprezintă, de asemenea, o codificare greșită pentru că începutul codificării pentru A este identic cu codificarea pentru C (o secvență precum ––···· este ambiguă, putând însemna fie AB, fie CC).

Se știe că într-o transmisie telegrafică, punctul durează o secundă, iar linia două secunde. Putem calcula astfel timpul necesar transmiterii unui text.

Folosind codificarea din primul exemplu, transmiterea textului $CABCA$ = –·–·····–· durează 11 secunde. Observați că lungimea transmisiei se poate calcula și astfel: $2 \cdot (A) + 1 \cdot (B) + 2 \cdot (C) = 2 \cdot (\cdot\cdot) + 1 \cdot (\cdot–) + 2 \cdot (–) = 2 \cdot 2 + 1 \cdot 3 + 2 \cdot 2 = 11$.

Se consideră un text, dat prin frecvența apariției fiecărui simbol (dintre cele 36 considerate). Să se găsească durata minimă necesară transmiterii acelui text, folosind o codificare aleasă corespunzător.

Date de intrare

Fișierul **telegraf.in** conține o singură linie cu 36 de numere întregi nenegative, separate prin câte un spațiu, re-

prezentând numărul de apariții ale fiecărui simbol în textul care trebuie transmis.

Date de ieșire

Fișierul **telegraf.out** va conține un singur număr, și anume lungimea minimă (în secunde) necesară pentru transmiterea textului.

Restricții și precizări

- nici unul din cele 36 de simboluri nu apare de mai mult de 1.000.000 de ori în textul considerat;
- există cel puțin două simboluri cu număr de apariții nenul.

Exemplu

telegraf.in

```
212000000000000000000000000000000000
```

telegraf.out

```
11
```

Timpi de execuție:

- *Microsoft Windows*: 0,3 secunde/test
- *Linux*: 0,1 secunde/test

P050334: Ajutor

Dacă te-ai rănit, nu folosi *Sprite* să tratezi rana. Cel mai bine e să fugi la cel mai apropiat post de prim ajutor. Norocul tău este că ai o hartă din care poți afla coordonatele carteziene ale posturilor de prim ajutor. Ghinionul este că, din cauza durerii, nu poți să fugi direct spre un post, ci numai pe direcțiile nord-sud și est-vest. Ca să știi dacă mai are sens să fugi spre un post sau să te bucuri de ultimele clipe de viață, cel mai bine e să evaluezi distanța până la cel mai apropiat post de prim ajutor. Cel mai apropiat post este cel pentru care distanța *Manhattan* este minimă.

De data aceasta ai scăpat pentru că timpul necesar ajungerii până la post a fost suficient; însă îți pui întrebarea: dacă accidentul s-ar fi întâmplat în alt loc, ai fi scăpat?

Se consideră N puncte în plan, reprezentând posturile de prim ajutor și alte M puncte reprezentând posibile locații ale accidentului. Se cere pentru fiecare dintre cele M puncte distanța *Manhattan* până la cel mai apropiat post.

Date de intrare

Fișierul **ajutor.in** conține pe prima linie numerele întregi N și M, în această ordine, separate printr-un spațiu. Pe următoarele N linii se află câte două numere întregi, separate printr-un spațiu, reprezentând coordonatele fiecărui post de prim ajutor. Pe următoarele M linii se află câte două numere întregi, separate printr-un spațiu, reprezentând coordonatele punctelor de accident. Coordonatele se dau în ordinea: abscisă, ordonată.

Date de ieșire

Fișierul **ajutor.out** va conține M linii, cu câte un număr pe fiecare linie, reprezentând distanța minimă până la cel mai apropiat post de prim ajutor.

Restricții și precizări

- $0 < N < 401$;
- $0 < M < 500.001$;
- toate coodonatele sunt numere întregi cuprinse în intervalul [0, 32000];
- dacă te afli deja la un post de prim ajutor (coordonatele sunt identice) distanța este 0;
- distanța *Manhattan* este cel mai scurt drum între două puncte, mergând doar pe direcții paralele cu axele de coordonate, adică $|x_1 - x_2| + |y_1 - y_2|$, unde (x_1, y_1) și (x_2, y_2) sunt coordonatele celor două puncte;
- pot exista puncte cu aceleași coordonate.

Exemplu

ajutor.in

```
4 4
1 1
5 5
1 5
5 1
0 0
1 1
3 3
4 1
```

ajutor.out

```
2
0
4
1
```

Timpi de execuție:

- *Microsoft Windows*: 2 secunde/test
- *Linux*: 1,3 secunde/test

P050335: Ghizi

Se caută ghizi pentru *Olimpiada Națională de Informatică*. Deoarece la *Olimpiadă* participă K echipe, trebuie ca în fiecare moment de timp din intervalul [0, 100) să existe exact K ghizi.

Pentru posturile de ghid s-au înscris N voluntari, care au fost numerotați cu numere distincte cuprinse între 1 și N. Fiecare dintre cei N voluntari a specificat un interval de timp în care poate asigura servicii de ghid. Voluntarul i poate fi ghid în intervalul $[T1_i, T2_i)$ (intervalul este închis în $T1_i$ și deschis în $T2_i$).

Dându-se intervalele de timp asociate celor N voluntari, determinați o variantă de angajare, astfel încât în fiecare moment de timp să fie prezenți exact K ghizi. Numărul total de voluntari angajați este irelevant.

Date de intrare

Prima linie a fișierului de intrare **ghizi.in** conține două numere întregi N și K, separate printr-un spațiu, cu semnificațiile de mai sus. Voluntarii sunt numerotați distinct, de la 1 la N. Fiecare dintre următoarele N linii conține descrierea unui voluntar; mai exact linia $i + 1$ conține valorile întregi $T1_i$ și $T2_i$ pentru voluntarul i.

Date de ieșire

În fișierul **ghizi.out** veți afișa pe prima linie numărul total al ghizilor angajați (M). Pe a doua linie veți scrie M numere distincte între 1 și N, ordonate crescător, reprezentând numerele de ordine ale ghizilor angajați.

Restricții și precizări

- $1 \le N \le 5.000$;
- $0 \le T1 \le T2 \le 100$ pentru fiecare dintre cei N voluntari;
- $0 \le K \le N$;
- pot exista doi voluntari cu același interval asociat;
- întotdeauna va exista soluție;
- dacă există mai multe soluții, se va afișa una oarecare.

Exemplu

ghizi.in	ghizi.out
6 2 0 100 0 15 15 99 0 10 10 20 20 100	4 1 4 5 6

Timpi de execuție:

- *Microsoft Windows*: 0,3 secunde/test
- *Linux*: 0,1 secunde/test

P050336: Sala

Președintele unui *Concurs de Informatică* dorește ca la festivitatea de încheiere să fie o atmosferă plăcută. Pentru acest lucru el vrea să așeze participanții pe n rânduri după anumite reguli. Aceste reguli sunt:

- pe primul rând al sălii așează una lângă alta n persoane;
- pe rândurile următoare așezarea pe poziția i (numerotarea se face de la stânga la dreapta) este condiționată de așezarea persoanelor de pe rândul anterior, adică dacă pe rândul din față pe pozițiile i și $i + 1$ stau fie numai băieți fie numai fete, atunci se va așeza o fată, iar dacă pe aceste poziții stau persoane de sex opus se va așeza un băiat.

Conform acestei reguli pe rândul cu numărul de ordine i ($i \in \{1, 2, ..., n\}$) se vor așeza $n - i + 1$ persoane.

Pentru n dat se cere să se determine numărul maxim de băieți care pot lua loc în sală, astfel încât să se respecte regulile anterioare.

Date de intrare

În fișierul de intrare **sala.in** pe prima linie se află numărul de cifre ale lui n, iar pe linia a doua se află cifrele numărului n separate între ele prin câte un spațiu.

Date de ieșire

În fișierul de ieșire **sala.out** pe prima linie se va afișa numărul din cerință.

Restricție

- $1 \le n \le 10^{101}$.

Exemple

sala.in	sala.out
1 5	10

sala.in	sala.out
2 1 3	61

Timp de execuție: 0,1 secunde/test

Câștigătorii

Clasa a V-a

Alexandru-Stan Mile, Timișoara - premiul I
Mihail Drăghici, Ploiești - premiul II
Robert Asaftei, Vaslui - premiul III

Clasa a VI-a

Cotizu Sima, Ploiești - premiul I
Adela Lica, Ploiești - premiul II
Ana Moisin, București - premiul III

Clasa a VII-a

Filip Buruiana, Galați - premiul I
Mihai Morariu, București - premiul II
Adela Rădoi, Drobeta Turnu-Severin - premiul II

Clasa a VIII-a

Valentin Stanciu, București - premiul I
Sorin Dascălu, Piatra Neamț - premiul II
Andrei Ioniță, Suceava - premiul II

Clasa a IX-a

Mircea Pașoi, Ploiești - premiul I
Dan Spătărel, București - premiul II
Radu Marin, Focșani - premiul III

Clasa a X-a

Sorin Stancu-Mara, București - premiul I
Daniel Fechete, Suceava - premiul II
Leonard Crestez, Brăila - premiul II
Gabriel Rizuc, Iași - premiul III
Korodi Andor-Csaba, Târgu Mureș - premiul III

Clasa a XI-a

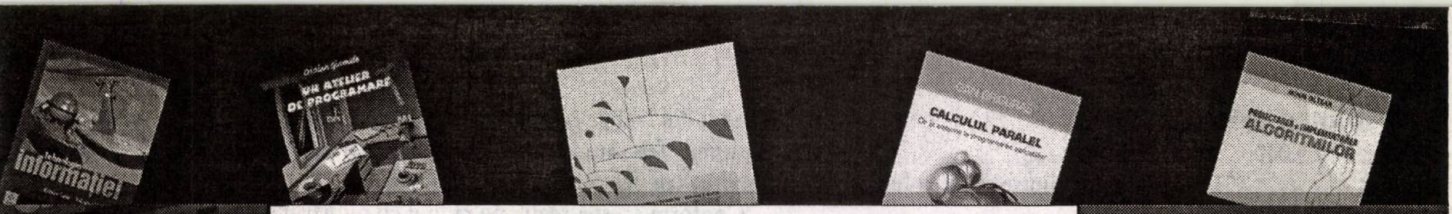
Silviu Gânceanu, Vaslui - premiul I
Emilian Miron, Bacău - premiul I
Andrei Ion, Constanța - premiul II
Mircea Digulescu, București - premiul III
Dan Ghinea, București - premiul III

Clasa a XII-a

Radu Berinde, București - premiul I
Tiberiu Dănet, București - premiul II
Andrei Giurgiu, București - premiul III

Membrii Lotului Național de Informatică:

Costel-Cătălin Antohi, Galați
Radu Berinde, București
Ștefan Ciobâcă, Suceava
Victor Costan, București
Leonard Crestez, Brăila
Tiberiu Dănet, București
Mircea Digulescu, București
Constantin Dumitrașcu, Focșani
Dan-Ionuț Fechete, Suceava
Ștefan Gheorghe, București
Dan Ghinea, București
Matei Gruber, București
Mihnea Giurgea, Ploiești
Andrei Homescu, Gorj
Andrei-Gabriel Ion, Constanța
Emilian Miron, Bacău
Alexandru Mosoi, București
Marius Nicolae, Slatina
Mircea Pașoi, Ploiești
Iolanda Popa, Iași



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 192422

COMPUTER LIBRIS AGORA

Rundele #17 - #20

Vom prezenta în cadrul acestui articol câteva informații referitoare la desfășurarea ultimelor patru runde ale concursului de programare organizat de revista noastră. De asemenea, vom prezenta clasamentul final, precum și lista concurenților calificați la etapa finală.

P040315: Heroes of Might and Magic

Primul perimetru de apărare este dat de înfășurătoarea convexă a mulțimii de puncte care reprezintă coordonatele castelelor. Al doilea perimetru reprezintă înfășurătoarea convexă a punctelor rămase după eliminarea celor care se află pe primul perimetru. Al treilea perimetru reprezintă înfășurătoarea convexă a punctelor rămase după eliminarea celor care se află pe primele perimetre. Procedeul continuă până în momentul în care pentru toate castelele au fost stabilite perimetrele din care fac parte.

Așadar, această problemă este, de fapt, ***problema cepei.*** Datorită faptului că fiecare perimetru (eventual cu excepția ultimului) va conține cel puțin trei castele, numărul total al perimetrelor va fi cel mult egal cu o treime din numărul punctelor. Ordinul de complexitate al operației de determinare a înfășurătorii convexe este $O(N \cdot \log N)$; vom determina $O(N)$ astfel de perimetre, deci ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) \cdot O(N \cdot \log N) = O(N^2 \cdot \log N)$.

Algoritmul de determinare a înfășurătorii convexe este clasic. Îl puteți găsi la pagina 769 a cărții ***Introducere în algoritmi***, scrisă *Thomas H. Cormen, Charles E. Leiserson* și *Ronald R. Rivest*, a cărei traducere în limba română a fost publicată la editura *Computer Libris Agora* din *Cluj-Napoca.*

Pentru această problemă am primit 22 de soluții pentru care a fost acordat punctajul maxim. Concurenții care au primit 100 de puncte la această etapă a concursului nostru sunt: *Adrian Cârcu (Bistrița), Andrei Homescu (Târgu Jiu), Bogdan Nicolae (București), Ciprian Cană (Vaslui), Constantin Răuțu (Botoșani), Costel-Cătălin Antohi (Tecuci), Csaba Pătcaș (Cluj-Napoca), Dan-Ionuț Fechete (Suceava), Eugen Sorbalo (Chișinău - Republica Moldova), Florin Antonescu (Băicoi), Ioan Stan (Aiud), Ioan Sucan (Arad), Ionuț-Bogdan Petre (Plopeni), Ivan Kazmenko (Sankt Petersburg - Federația Rusă), Marius Nicolae (Slatina), Maximilian Machedon (București), Mugurel-Ionuț Andreica (București), Răzvan Albu (București), Silvestru Cosmin Negrușeri (Bistrița), Slotta Johannes (Jena - Republica Federală Germania), Ștefan Ciobâcă (Suceava),* și *Tiberiu Chiorean (Baia Mare).*

Numărul total al concurenților care au trimis soluții pentru această problemă a fost 45; așadar, aproximativ jumătate dintre ei au rezolvat-o corect.

P040316: Fotbal

Problema de la această rundă a fost ultimul "concurs de euristici" din acest an. Parțial, problema se putea rezolva cu ajutorul unui algoritm de determinare a cuplajului într-un graf. Astfel se puteau "repartiza" golurile în așa fel încât să se respecte cerințele din clasament pentru toate echipele, ceea ce ar fi dus la obținerea a 60% din punctaj.

Tot un algoritm de determinare a cuplajului poate fi folosit pentru ca să fie respectate condițiile referitoare la numărul de victorii, partide egale și înfrângeri. Acesta ar fi dus la obținerea a 40% din punctaj.

Din nefericire, cei doi algoritmi nu pot fi "combinați", așadar era necesară găsirea celui mai bun "compromis".

Din nou, câștigătorul acestui concurs a fost *Adrian Cârcu* din *Bistrița*, care a obținut în final 99,91 puncte (la nouă dintre cele douăzeci de teste el a "pierdut" 0,01 puncte; pentru celelalte clasamentul obținut de el a fost identic cu cel dat).

Doar 11 dintre concurenți au reușit să obțină cel puțin jumătate din punctajul maxim. Aceștia sunt:

1.	***Adrian Cârcu, Bistrița***	***100***
2.	***Ivan Kazmenko, Federația Rusă***	***99***
2.	***Mugurel-Ionuț Andreica, București***	***99***
4.	***Constantin Răuțu, Botoșani***	***98***
5.	***Csaba Pătcaș, Cluj-Napoca***	***96***
6.	***Tiberiu Chiorean, Baia Mare***	***95***
7.	***Aurelian Marinescu, Brașov***	***91***
8.	***Daniel Nechifor, Vaslui***	***88***
9.	***Bogdan Nicolae, București***	***78***
10.	***Dan Ghinea, București***	***65***
11.	***Cheng Sheng, Republica Populară Chineză***	***63***

La această rundă au fost evaluate soluțiile trimise de 26 dintre concurenții înscriși la această a patra ediție a con-

cursului de programare *Bursele Agora*.

P040317: Atlantis

Problema se reduce la determinarea numărului de submulțimi de sumă dată a unei mulțimi. Pentru cazul general, problema este *NP-completă*, dar în situații particulare pot fi găsiți algoritmi cu timp de execuție pseudopolinomial. Datorită faptului că fișierele de intrare erau deschise, numărul de elemente, suma căutată și valoarea rezultatului au fost destul de mari. Ca urmare, pentru ultimele patru teste nici unul dintre concurenți nu a reușit să găsească rezultatul corect. Până la urmă, redacția *GInfo* a decis să anuleze aceste teste.

Cu toate acestea, un singur concurent a obținut punctajul maxim (*Silvestru-Cosmin Negrușeri* din *Bistrița*), el reușind să determine cele 16 fișiere de ieșire corecte. Alți cinci concurenți au reușit să determine 15 dintre cele 16 fișiere de ieșire corecte, obținând 92 de puncte. Aceștia sunt: *Adrian Cârcu (Bistrița), Bogdan Nicolae (București), Csaba Pătcaș (Cluj-Napoca), Mugurel-Ionuț Andreica (București)* și *Ștefan Ciobâcă (Suceava)*. Ei au fost urmați de *Ivan Kazmenko* din *Federația Rusă* care a determinat 14 dintre fișierele corecte, obținând 84 de puncte. Următorii clasați au fost *Cristian Cernat (Târgu Jiu), Florin Antonescu (Băicoi)* și *Ionuț-Bogdan Petre (Plopeni)* cu câte 13 fișiere de ieșire corecte; ei au obținut câte 76 de puncte.

Numărul concurenților care au trimis fișiere de ieșire pentru această problemă a fost 27.

P040318: Maxime

În acest moment nu se cunoaște un algoritm optim (din punct de vedere al numărului de comparații efectuate) pentru determinarea pozițiilor celor mai mari trei numere dintr-un șir.

Problema determinării celor mai mari două numere a unui șir poate fi rezolvată optim, existând un algoritm care folosește $N + [\log_2 N] - 2$ comparații.

Acesta constă în construirea unui arbore ale cărui frunze sunt numerele șirului (în cazul nostru pozițiile). Se aleg două frunze și pe nivelul superior este plasată poziția celui mai mare dintre ele. Pentru acest nivel se aplică același algoritm și se continuă procedeul până în momentul în care rămânem cu un singur nod. Acesta reprezintă maximul și determinarea sa necesită $N - 1$ comparații.

Al doilea element ca mărime este unul dintre cele cu care a fost comparat maximul. Așadar, determinarea sa necesită efectuarea unui drum de la rădăcina arborelui la o frunză, deci necesită $[\log_2 N] - 1$.

Al treilea element este unul dintre cele cu care a fost comparat maximul sau al doilea element ca mărime, deci vom avea nevoie de încă două drumuri spre frunze, care vor necesita cel mult $2 \cdot [\log_2 N] - 2$ comparări.

Algoritmul descris nu este optim, dar utilizarea sa ar fi dus la obținerea punctajului maxim.

Problema s-a dovedit relativ simplă, 16 dintre cei 32 de concurenți care au trimis soluții au obținut punctajul maxim. Aceștia sunt: *Adrian Cârcu (Bistrița), Aurelian Marinescu (Brașov), Cătălin-Valeriu Liță (Bârlad), Călin Bindea (Bistrița), Ciprian Cană (Vaslui), Constantin Răuțu (Botoșani), Csaba Pătcaș (Cluj-Napoca), Dan Ghinea (București), Dumitru Codreanu (Chișinău - Republica Moldova), Iuliu Konya (Baia Mare), Maximilian Machedon (București), Mugurel-Ionuț Andreica (București), Silvestru-Cosmin Negrușeri (Bistrița), Sorin Stancu-Mara (București), Ștefan Ciobâcă (Suceava)* și *Tiberiu Chiorean (Baia Mare)*.

Etapa finală

Etapa finală a ediției din acest an se va desfășura pe data de ***17 mai 2003*** la *Liceul "Emil Racoviță"* din *Cluj-Napoca*.

Enunțurile celor patru probleme care vor fi propuse spre rezolvare vor fi publicate pe *site*-ul revistei noastre în dimineața zilei desfășurării concursului.

Cei care doresc ca soluțiile lor să fie evaluate, pot trimite rezolvările considerând că etapa finală este runda #21 a concursului.

Cele patru probleme propuse spre rezolvare vor face parte din cele patru categorii de probleme propuse pe parcursul desfășurării celor 20 de etape ale primei faze a concusrsului: o problemă "clasică", o problemă tip competiție, o problemă cu fișierele de intrare deschise și o problemă interactivă.

În urma desfășurării celor 20 de etape ale primei faze, pentru etapa finală s-au calificat următorii concurenți:

Adrian Cârcu, Bistrița
Mugurel-Ionuț Andreica, București
Csaba Pătcaș, Cluj-Napoca
Ivan Kazmenko, Federația Rusă
Dan Ghinea, București
Silvestru-Cosmin Negrușeri, Bistrița
Bogdan Nicolae, București
Ștefan Ciobâcă, Suceava
Aurelian Marinescu, Brașov
Tiberiu Chiorean, Baia Mare

Redacția *GInfo* mulțumește tuturor participanților pentru interesul acordat concursului *Bursele Agora*.

Ne exprimăm speranța că acest concurs s-a dovedit util și și-a atins scopul declarat: pregătirea elevilor și studenților în vederea participării la diferite concursuri de programare.

Cea de-a cincea ediție a concursului organizat de revista noastră va începe în toamna acestui an. V-am pregătit deja câteva surprize, sperăm noi plăcute, care vor face ca noua ediție să fie mai dinamică, mai interactivă, mai interesantă.

Noul regulamentul va fi publicat pe *site*-ul concursului (**www.ginfo.ro/concurs**) în cursul verii. Vor exista o mulțime de modificări; așadar vă recomandăm să îl citiți cu atenție înainte de a vă înscrie la concurs.

Deocamdată nu este totul stabilit, motiv pentru care, pe parcursul acestei veri, așteptăm pe adresa redacției opi-

Clasament FINAL

1. Adrian Cârcu, Bistrița 1963
2. Mugurel-Ionuț Andreica, București 1892
3. Csaba Pătcaș, Cluj-Napoca 1746
4. Ivan Kazmenko, Sankt Petersburg - Federația Rusă 1720
5. Dan Ghinea, București 1667
6. Silvestru-Cosmin Negruşeri, Bistrița 1661
7. Bogdan Nicolae, București 1606
8. Ştefan Ciobâcă, Suceava 1577
9. Aurelian Marinescu, Braşov 1574
10. Tiberiu Chiorean, Baia Mare 1498

11. Ciprian Cană, Vaslui 1497
12. D. Codreanu, R. Moldova 1472
13. Daniel Nechifor, Vaslui 1404
14. M. Machedon, București 1296
15. Iuliu Konya, Baia Mare 1282
16. Ionuț-Bogdan Petre, Plopeni 1214
17. Răzvan Albu, București 1208
18. Marius Nicolae, Slatina 1197
19. Iulian Blajut, Bacău 1190
20. Dan-Ionuț Fechete, Suceava 1169
21. Constantin Răuțu, Botoșani 1137
22. Andrei Homescu, Târgu Jiu 1124
23. Lucian Suditu, Vaslui 1059
24. Florin Antonescu, Băicoi 971
25. Radu Vatavu, Suceava 875
26. Cristian Bojinovici, București 857
27. Cornel Mihăilă, Braşov 837
28. Alin Coman, Târgu Jiu 828
29. Mihail Popa, București 802
30. Călin Bindea, Bistrița 762
31. Johannes Slotta, Germania 708
32. C-tin Jucovschi, R. Moldova 698
33. Andrei G. Ion, Constanța 673
34. Andrei Gönczi, Arad 587
35. Adrian Nasui, Slobozia 585
36. Mircea Pașoi, Ploiești 569
37. Eugen Sorbalo, R. Moldova 527
38. Marius Airinei, Bacău 524
39. Silviu Hagea, Călărași 518
40. Marius-Dorin Moraru, Buftea 497
41. Andrei Blaj, Suceava 496
42. Alexandru Sofronia, Bacău 458
43. Alexandru Vlajoaga, Pitești 431
44. Cătălin-Valeriu Liță, Bârlad 428
45. Victor Costan, București 46. 397
46. Marian Raica, Drăgutești 374
47. Mihaela Marin, Codlea 369
48. Mohamed Taha, Egipt 367
49. M. Piț-Rada, Dr. Tr. Severin 364
50. Cosmin Molea, Bascov 353
50. Ștefăniță Fechete, Suceava 353
52. Radu Ștefan, Brașov 352
53. Adrian Balaşko, Zalău 349
54. Costel-Cătălin Antohi, Tecuci 336
55. Bogdan Pănuș, Satu Mare 334
56. Paul H. Stan, Cluj-Napoca 324
56. Radu Berinde, București 324
58. Mihai Stanimir, Bumbești-Jiu 316
59. Asif-ul Haque, Bangladej 311
60. Ágota Turánicz, Cluj-Napoca 307
61. Lego Haryanto, S.U.A. 306
61. Marius Dragomir, Alba Iulia 306
63. Robert-Emanuel Nica, Deleni 305
64. Andrei Csibi, Cluj-Napoca 300
64. Gabriel Rudeanu, Iași 300
66. Valentin Bran, Satu Mare 277
67. Cristian Cernat, Târgu Jiu 275
68. Ioan Stan, Aiud 274
69. Victor Podeanu, București 273
70. Alexandru Tomescu, București 260
71. Alexandru Moșoi, Bacău 256
72. Csaba Andraș, Oradea 246
73. Alex Florescu, București 236
74. Angelo Lupuț, Satu Mare 235
75. Vlad Dascălu, Bacău 226
76. Răzvan Tigaeru, Călărași 220
77. S. Spătaru, Zimandu Nou 217
78. Călin Ciutu, București 214
79. Horea Cărămizaru, Craiova 212
80. Andrei Păduraru, Vaslui 200
80. Sorin Bălășioiu, Craiova 200
80. S. Stancu-Mara, București 200
83. Alex. Constantin, Pitești 196
84. Alexandru Stan, Ploiești 193
85. Silviu Gânceanu, Bârlad 179
86. Ralph Bottesch, Sibiu 166
87. Alex. Constantin, Ploiești 164
88. Saratov Team, Rusia 155
89. Ștefan Bucur, București 155
90. Natalia Baranova, Rusia 154
91. A.G. Matei, Câmpulung M. 147
92. Daniel Păsăilă, Suceava 144
93. Bogdan Chirițoiu, Ploiești 143
94. Răzvan Bradea, Galați 141
95. Vincent Groenhuis, Olanda 137
96. Alpár F. Perini, Sf. Gheorghe 131
97. Vasile Răcășan, Codlea 130
98. Emilian Miron, Bacău 125
99. Vlad Bălan, Bacău 120
100. M.O. Dâmboiu, București 117

BILANȚ

Prima fază a celei de-a patra ediții a concursului internațional de programare Bursele Agora s-a încheiat. În cadrul acestui articol vom prezenta câteva date statistice referitoare la desfășurarea acestei ediții a concursului.

Concurenții

La ediția a patra a concursului de programare organizat de revista noastră s-au înscris 476 de concurenți. Datorită nerespectării anumitor prevederi ale regulamentului, 12 concurenți au fost descalificați. Dintre cei 464 de concurenți rămași, doar 189 au reușit să obțină cel puțin un punct la concursul nostru.

Marea majoritate a concurenților au fost români, dar au participat și 56 de concurenți din afara granițelor țării. Am avut concurenți din 21 de țări ale lumii: *Bangladej*, *Brazilia*, *Canada*, *China*, *Columbia*, *Egipt*, *Filipine*, *Iugoslavia*, *Marea Britanie*, *Germania*, *Olanda*, *India*, *Mauritus*, *Moldova*, *Polonia*, *România*, *Rusia*, *Statele Unite*, *Suedia*, *Ucraina* și *Vietnam*.

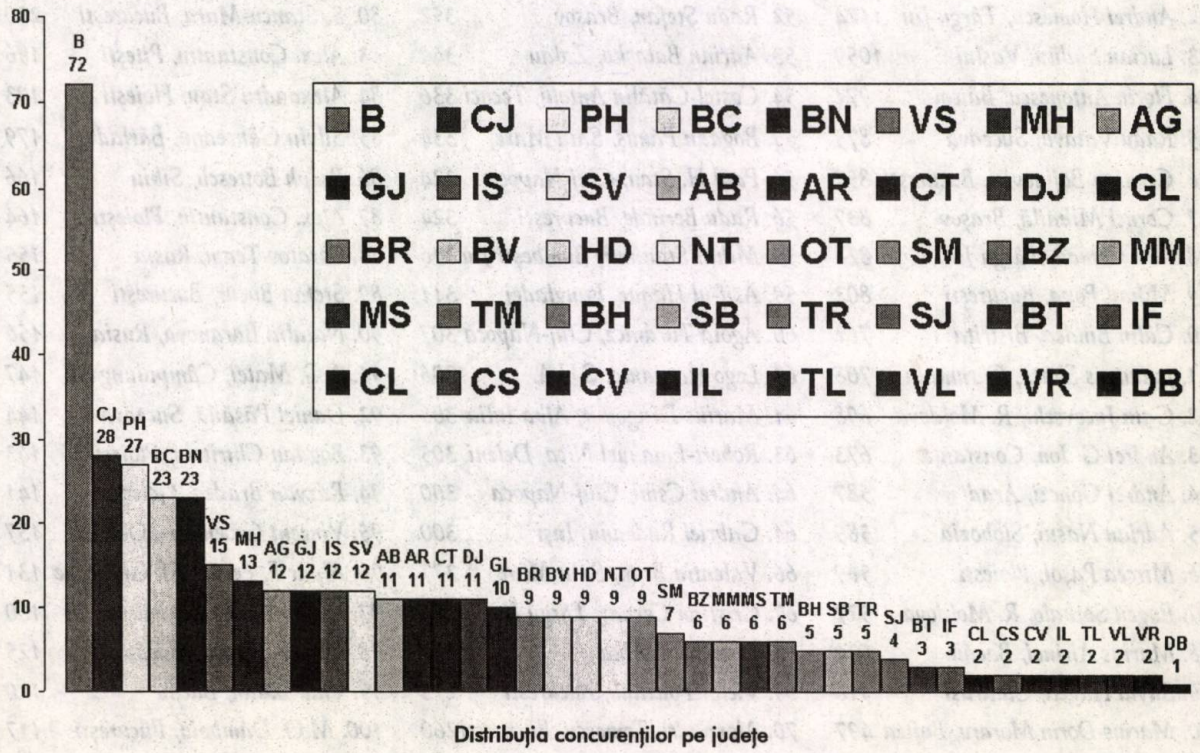
La această ediție au participat elevi și studenți din aproape toate județele țării (au lipsit doar *Giurgiu* și *Harghita*). În graficul de pe această pagină este prezentată statistica repartizării concurenților pe județele țării.

Performanțe

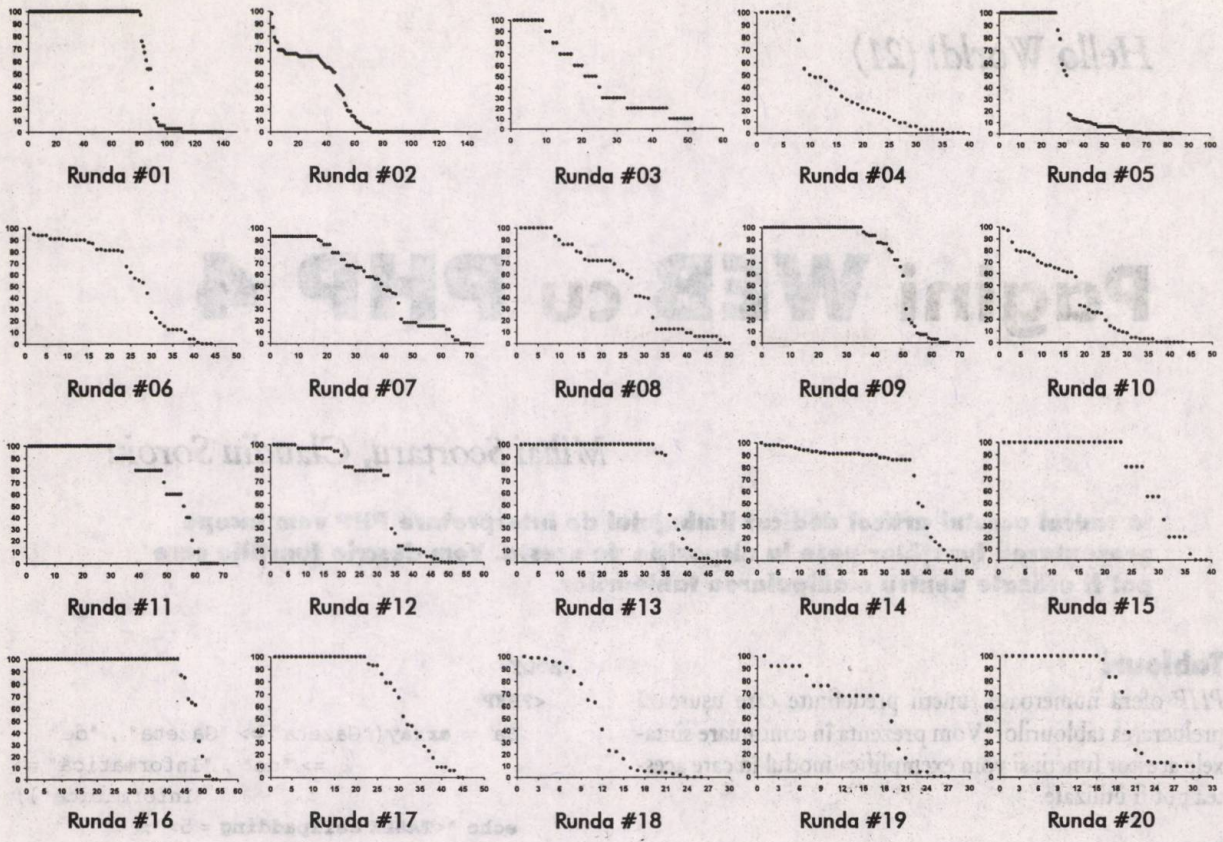
Dintre participanți, doar 23 au reușit să obțină cel puțin 1000 dintre cele 2000 de puncte puse în joc. Așadar, numărul concurenților care au reușit să obțină cel puțin jumătate din punctajul maxim este relativ mic, ceea ce demonstrează că nivelul de dificultate al acestui concurs a fost destul de ridicat.

Totuși, 121 dintre cei care au trimis soluții au reușit să obțină punctajul maxim la cel puțin una dintre cele 20 de probleme.

Concurentul care a rezolvat integral cele mai multe probleme a fost *Adrian Cârcu* din *Bistrița*. El a reușit să obțină punctajul maxim la 15 dintre cele 20 de runde.



Distribuția concurenților pe județe



Distribuția punctajelor la cele 20 de runde

Problemele

Au fost 20 de runde grupate în patru categorii: probleme clasice, probleme de tip competiție, probleme cu fișierele de intrare deschise și probleme interactive; din fiecare categorie am ales cinci probleme.

Au apărut mult mai puține probleme decât ne-am așteptat; nu au existat întârzieri la publicarea rezultatelor sau a enunțurilor, chiar dacă "agenda" a fost destul de încărcată (a fost organizată câte o rundă în fiecare săptămână cu excepția perioadei sărbătorilor de iarnă, când concursul a fost întrerupt și cea a sesiunii de iarnă când am avut câte o rundă la fiecare două săptămâni).

Datorită ritmului alert în care s-a desfășurat competiția, nu am mai avut posibilitatea de a accepta contestații. Din acest punct de vedere, toți concurenții au fost egali. Punctajele acordate nu au fost modificate decât în cazuri excepționale, cum ar fi erorile apărute în momentul introducerii datelor sau situațiile în care, pur și simplu, am sărit peste sursele unor concurenți.

Totuși, desfășurarea nu a fost perfectă. La două dintre problemele la care fișierele de intrare au fost deschise, unele dintre fișierele pe care vi le-am pus la dispoziție nu erau corecte. Ne cerem încă o dată scuze pentru aceste evenimente neplăcute.

Unele probleme au fost mai dificile, altele au fost foarte simple. În graficele de pe această pagină este prezentată configurația punctajelor la fiecare dintre cele 20 de runde (au fost luați în considerare doar concurenții care nu au fost descalificați ulterior și care au trimis soluții care au putut fi evaluate).

O estimare aproximativă a dificultății problemelor rezultă din forma graficelor. Potrivit acestui criteriu, cea mai simplă problemă clasică a fost cea de la runda #13, cea mai simplă problemă de tip competiție a fost cea de la runda #14, cea mai simplă problemă cu fișierele de intrare deschise a fost cea de la runda #15, iar cea mai simplă problemă interactivă a fost cea de la runda #16.

Potrivit aceluiași criteriu, cea mai dificilă problemă clasică a fost cea de la runda #05, cea mai dificilă problemă de tip competiție a fost cea de la runda #02, cea mai dificilă problemă cu fișierele de intrare deschise a fost cea de la runda #03, iar cea mai dificilă problemă interactivă a fost cea de la runda #04.

Aceste estimări se bazează doar pe distribuția punctajelor obținute de concurenți și s-ar putea să nu reflecte exact realitatea.

Etapa finală

Etapa finală din acest an a concursului de programare *Bursele Agora* va avea loc pe data de 17 mai. Enunțurile vor fi disponibile *online* pe *site*-ul revistei noastre în momentul începerii concursului. Tot pe *site* va fi anunțată data la care va avea loc finala din acest an. Vor fi patru probleme (câte una din fiecare categorie), clasamentul final fiind stabilit pe baza punctajelor obținute la aceste probleme și, eventual, pe baza punctajului de la cele 20 de runde, în caz de egalitate.

Pagini WEB cu PHP 4

Mihai Scorţaru, Claudiu Soroiu

În cadrul acestui articol dedicat limbajului de interpretare PHP vom începe prezentarea funcţiilor puse la dispoziţie de acesta. Vom descrie funcţiile care pot fi utilizate pentru manipularea tablourilor.

Tablouri

PHP oferă numeroase funcţii predefinite care uşurează prelucrarea tablourilor. Vom prezenta în continuare sintaxele acestor funcţii şi vom exemplifica modul în care acestea pot fi utilizate.

Funcţia array_change_key_case

Această funcţie realizează conversia tuturor literelor care apar în cadrul denumirilor cheilor elementelor unui tablou.

Funcţia are doi parametri: un tablou asupra căruia se vor efectua modificările şi un întreg care arată dacă se vor converti toate literele mari în litere mici sau invers.

Pentru acest parametru pot fi utilizate constantele predefinite CASE_UPPER (conversie la litere mari) şi CASE_LOWER (conversie la litere mici).

Acest parametru poate lipsi, caz în care va avea valoarea implicită CASE_LOWER, deci va realiza conversia tuturor literelor mari în litere mici.

Următorul document *PHP* ilustrează modul în care poate fi utilizată această funcţie. Efectul este prezentat în figura 1.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcţia array_change_key_case
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $a = array("Gazeta" => "Gazeta" , "de"
                          => "de" , "Informatică" =>
                                "Informatică");
      echo "<TABLE cellpadding = 5>";
      echo "<TR><TD><B>Tabloul iniţial:"
      echo "</B><TD><TT>";
      print_r($a);
      echo "</TT><TR><TD><B>Tabloul după "
      echo "conversia implicită:"
      echo "</B><TD><TT>";
      $b = array_change_key_case($a);
      print_r($b);
      echo "</TT><TR><TD><B>Tabloul după "
      echo "conversia la litere mari:"
      echo "</B><TD><TT>";
      $b = array_change_key_case($a,CASE_UPPER);
      print_r($b);
      echo "</TT><TR><TD><B>Tabloul după "
      echo "conversia la litere mici:"
      echo "</B><TD><TT>";
      $b = array_change_key_case($a,CASE_LOWER);
      print_r($b);
    echo "</TT></TABLE>"
    ?>
  </BODY>
</HTML>
```

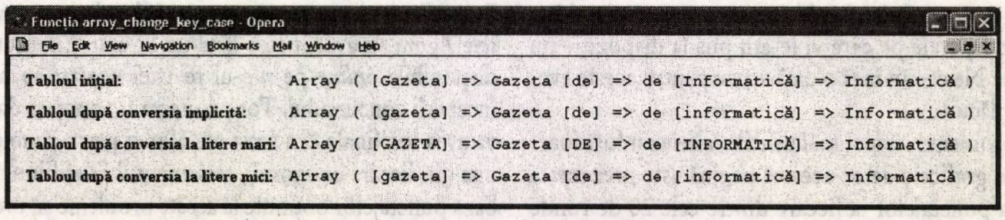


Figura 1: Funcţia array_change_key_case

După cum se poate vedea în figură, funcția are efect doar asupra cheilor elementelor, valorile acestora rămânând nemodificate.

Funcția array_chunks

Această funcție poate fi utilizată pentru a împărți un tablou în mai multe bucăți (*chunks*). Ea are trei parametri, și anume: vectorul care urmează a fi împărțit, dimensiunea unei bucăți (numărul de elemente) și un indicator care arată dacă se păstrează sau nu cheile elementelor (acesta poate lipsi; în acest caz are valoarea implicită **false**, deci nu se vor păstra cheile elementelor).

În cazul în care numărul elementelor tabloului inițial nu este multiplu al dimensiunii unei bucăți, atunci ultima bucată va fi mai "scurtă". Funcția va returna un tablou al bucăților (un tablou de tablouri).

Prezentăm în continuare un document *PHP* care ilustrează modul în care poate fi utilizată această funcție. Efectul este prezentat în figura 2.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția array_chunks
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $vector = array(
       "a","b","c","d","e");
      echo "<PRE>";
      print_r(array_chunk
              ($vector,2));
      echo "<P>";
      print_r(array_chunk(
           $vector,2,TRUE));
      echo "</PRE>";
    ?>
  </BODY>
</HTML>
```

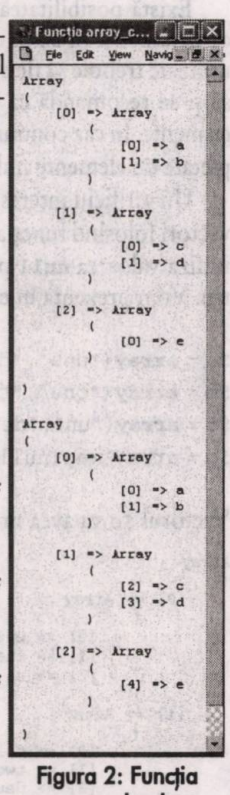


Figura 2: Funcția array_chunks

Funcția array_count_values

Această funcție are ca parametru un tablou și returnează un alt tablou ale cărui elemente au chei date de valorile elementelor tabloului transmis ca parametru. Pentru fiecare valoare se va genera un singur element (chiar dacă valoarea apare de mai multe ori), iar valoarea elementului respectiv este dată de numărul aparițiilor.

De exemplu, în urma execuției unei secvențe de forma:

```
$tablou = array(1,"hello",1,"world","hello");
$rez = array_count_values ($tablou));
```

vectorul `$rez` va conține trei elemente, acestea vor avea cheile `1`, `hello` și `world`, iar valorile vor fi 2, 2, respectiv 1.

Funcțiile array_diff și array_diff_assoc

Uneori dorim să eliminăm dintr-un tablou anumite elemente; în acest scop pot fi utilizate funcțiile `array_diff()` și `array_dif_assoc()`.

Acestea au ca parametri mai multe tablouri și returnează un tablou care conține toate elementele din tabloul reprezentat de primul parametru care nu apar în nici unul dintre tablourile corespunzătoare celui de-al doilea parametru.

În cazul funcției `array_diff()` sunt eliminate elemente din primul tablou dacă în celelalte există elemente cu aceeași valoare.

În cazul funcției `array_diff_assoc()` sunt eliminate elemente din primul tablou dacă în celelalte există elemente cu aceeași valoare și aceeași cheie.

Funcția array_fill

Această funcție poate fi utilizată pentru a completa cu o anumită valoare elementele unui tablou. Funcția are trei parametri: primul reprezintă indicele la care începe completarea, al doilea reprezintă numărul elementelor completate, iar cel de-al treilea indică valoarea cu care se completează elementele.

De exemplu, în urma executării instrucțiunii `$a = array_fill(5,6,'Hello World!');`, vom avea un tablou cu șase elemente (cheile acestora sunt `5`, `6`, `7`, `8`, `9` și `10`), valorile tuturor acestora fiind `Hello World!`.

Funcția array_filter

Această funcție realizează o filtrare a elementelor unui vector pe baza unei funcții. Funcția `array_filter()` are ca parametri un tablou și identificatorul funcției care este utilizată pentru filtrare.

Această funcție trebuie să fie definită și va fi apelată pentru fiecare element al vectorului. În cazul în care funcția returnează valoarea **true** (eventual, în urma conversiei la tipul `boolean`), elementul este păstrat; în caz contrar, acesta este eliminat.

Funcția `array_filter()` returnează vectorul obținut după eliminarea elementelor.

În continuare vom prezenta un exemplu în care se utilizează această funcție pentru a filtra elementele unui vector în funcție de paritatea acestora. Pagina *Web* generată este prezentată în figura 3.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția array_filter
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      function impar($var) {
        return $var % 2;
      }
```

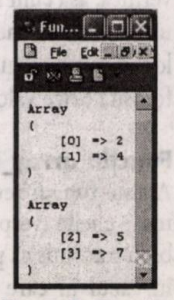


Figura 3: Funcția array_filter

```
    function par($var) {
      return 1 - $var % 2;
    }
    $vector = array("2","4","5","7");
    echo "<PRE>";
    print_r(array_filter($vector,par));
    echo "<P>";
    print_r(array_filter($vector,impar));
    echo "</PRE>";
  ?>
 </BODY>
</HTML>
```

Funcția array_flip

Această funcție are ca parametru un tablou și returnează un alt tablou în care a fost realizată o interschimbare a cheilor elementelor cu valorile. Cu alte cuvinte, pentru fiecare element va fi creat un element a cărui cheie va fi valoarea elementului în tabloul inițial și a cărui valoare va fi cheia elementului din tabloul inițial.

De exemplu, pentru tabloul definit prin

```
$tablou = array("A" => "Gazeta" , "B" => "de" ,
                                  "C" => "Informatică");
```

în urma executării apelului `$b = array_flip($tablou)` tabloul `$b` va avea forma `array("Gazeta" => "A" , "de" => "B" , "Informatică" => "C")`.

În cazul în care în vectorul transmis ca parametru există mai multe elemente care au aceeași valoare, datorită faptului că elementele unui vector trebuie să sibă chei distincte, vor fi eliminate toate aceste elemente, cu excepția ultimului.

Funcțiile array_intersect și array_intersect_assoc

Aceste funcții au un comportament similar funcțiilor `array_diff()`, respectiv `array_diff_assoc()`. Singura diferență este dată de faptul că tabloul rezultat va conține toate elementele care se află în fiecare dintre tablourile transmise ca parametri (pentru funcția `array_diff()` elementele trebuie să aibă aceeași valoare, în timp ce pentru funcția `array_diff_assoc()` ele trebuie să aibă aceeași cheie și aceeași valoare).

Funcția array_key_exists

Această funcție are doi parametri (o cheie și un vector) și verifică, așa cum sugerează și numele, dacă vectorul conține un element cu cheia respectivă. Funcția returnează o valoare logică, iar pentru parametrul care reprezintă cheia poate fi folosită orice valoare care poate fi cheie într-un vector.

Funcția array_keys și array_values

Aceste funcții returnează un vector care conține ca valori toate cheile (respectiv toate valorile) elementelor unui tablou transmis ca parametru. Pentru funcția `array_keys()`, în cazul în care se folosește un parametru suplimentar, vectorul rezultat va conține doar cheile elementelor care au valoarea egală cu cea indicată de acest parametru.

Funcția array_map

Această funcție realizează o transformare a elementelor unui vector pe baza unei funcții. Funcția este transmisă ca parametru și este aplicată tuturor elementelor unui vector care este, și el, transmis ca parametru.

Iată un exemplu:

```
function cub($n) {
  return $n*$n*$n;
}
$a = array(1, 2, 3, 4, 5);
$b = array_map("cub", $a);
```

În urma executării secvenței, vectorul `$b` va avea cinci elemente ale căror valori vor fi 1, 8, 27, 64 și 125.

Există posibilitatea de a transmite mai multe tablouri ca parametri. În acest caz, numărul argumentelor funcției utilizate trebuie să fie egal cu numărul tablourilor transmise și se recomandă ca tablourile să aibă același număr de elemente. În caz contrar, tablourile mai "scurte" vor fi completate cu elemente nule.

Un artificiu interesant permite crearea unor vectori de vectori folosind funcția `array_map()`. Pentru aceasta vom utiliza valoarea **null** pentru funcția transmisă ca parametru. Vom prezenta în continuare un exemplu:

```
$a = array("unu","doi","trei");
$b = array("one","two","three");
$c = array("un","deux","trois");
$d = array_map(null,$a,$b,$c);
```

Vectorul `$d` va avea următoarea structură:

```
Array
(
    [0] => Array
        (
            [0] => unu
            [1] => one
            [2] => un
        )
    [1] => Array
        (
            [0] => doi
            [1] => two
            [2] => deux
        )
    [2] => Array
        (
            [0] => trei
            [1] => three
            [2] => trois
        )
)
```

Funcțiile array_merge și array_merge_recursive

Aceste funcții realizează o "interclasare" a două sau mai multe tablouri prin adăugarea elementelor unui vector la sfârșitul tabloului anterior. În cazul în care există elemente cu aceeași cheie (reprezentată de un șir de caractere) în tabloul rezultat va apărea un singur element cu cheia respectivă, și anume primul întâlnit în timpul interclasării. În cazul în care cheia identică este reprezentată de un număr, în tabloul rezultat vor apărea ambele valori, dar valoarea corespunzătoare unei chei care apare deja în tablou va avea

cheia reprezentată de cel mai mic număr mai mare decât cheia inițială care nu apare încă în tabloul rezultat.

În cazul în care tablourile conțin elemente care sunt tablouri, atunci funcția este `array_merge_recursive()` și are ca efect și interclasarea acestor tablouri (care sunt elemente ale altor tablouri) și procedeul continuă recursiv. În cele ce urmează vom prezenta un document *PHP* care ilustrează diferențele dintre cele două funcții. Efectul este prezentat în figura 4.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcțiile
      array_merge și
      array_merge
      _recursive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $v1 = array(
      "culoare" =>
      array(
      "preferată" =>
      "roșu"),(5);
      $v2 = array(
      10,("culoare"
      =>(array(
      "preferată" =>
      "verde",
      "albastru"));
      echo "<PRE>";
      print_r(array_merge($v1,$v2));
      echo "<P>";
      print_r(array_merge_recursive($v1,$v2));
      echo "</PRE>";
    ?>
  </BODY>
</HTML>
```

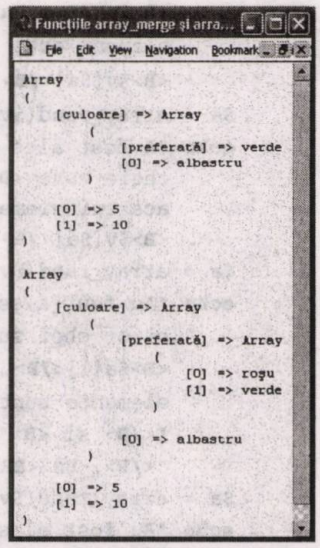


Figura 4: Funcțiile array_merge și array_merge_recursive

Funcția array_multisort

Această funcție poate fi utilizată pentru a sorta mai multe tablouri sau un tablou multidimensional.

Primul parametru al funcției trebuie să fie un tablou, iar următoarele pot fi tablouri și indicatori de sortare. Fiecare indicator de sortare descrie caracteristici ale sortării pentru cel mai apropiat tablou aflat înaintea indicatorului în lista de parametri.

Există două categorii de indicatori de sortare:

- indicatori de ordine:
  - SORT_ASC: sortare în ordine crescătoare;
  - SORT_DESC: sortare în ordine descrescătoare;
- indicatori de tip :
  - SORT_REGULAR: elementele sunt comparate fără a efectua conversii;
  - SORT_NUMERIC: elementele sunt comparate după ce sunt convertite la valori numerice;
  - SORT_STRING: elementele sunt comparate după ce sunt convertite în șiruri de caractere.

Pentru un tablou nu pot fi specificați doi indicatori care fac parte din aceeași categorie (de exemplu, nu este permisă specificarea pentru un tablou a indicatorilor SORT_NUMERIC și SORT_STRING).

Nu este obligatorie specificarea ambelor tipuri de indicatori pentru fiecare tablou. În cazul în care indicatorii lipsesc, sunt utilizate valorile implicite SORT_ASC și SORT_REGULAR.

Tablourile transmise ca parametri sunt considerate a fi coloane într-o matrice care trebuie sortată pe linii. Primul tablou reprezintă criteriul de bază. Pentru valori egale în primul tablou se ia în considerare al doilea criteriu (reprezentat de al doilea tablou). Pentru valori egale în primele două tablouri se ia în considerare al treilea criteriu și așa mai departe.

Funcția returnează o valoare logică ce indică dacă operația de sortare s-a încheiat cu succes.

Să considerăm doi vectori definiți astfel:

```
$v1 = array("10",100,100,"a");
$v2 = array(1,3,"2",1);
```

În urma execuției apelului `array_multisort($v1, $v2);` vectorul `$v1` va conține elementele `"10"`, `"a"`, `100` și `100`, în timp ce vectorul `$v2` va conține elementele `1`, `1`, `"2"` și `3`. Se observă că elementele din al doilea vector corespunzătoare elementelor cu valori egale din primul vector (`3` și `"2"` corespund celor două elemente cu valoarea `100` din primul vector) au fost și ele sortate.

Vom prezenta acum un exemplu în care vom sorta un tablou bidimensional definit prin:

```
$v = array(array("10",100,100,"a"),
           array(1,3,"2",1));
```

În urma unui apel de forma:

```
array_multisort($v[0],SORT_ASC,SORT_STRING,
                $v[1],SORT_NUMERIC,SORT_DESC);
```

prima linie a tabloului va conține elementele `"10"`, `100`, `100` și `"a"` (au fost sortate crescător ca șiruri de caractere), iar cea de-a doua va conține elementele `1`, `3`, `"2"` și `1` (pentru valori egale în prima linie, elementele de pe a doua au fost sortate descrescător ca valori numerice).

Funcția array_multisort

Această funcție poate fi utilizată pentru a completa un tablou cu o valoare dată, astfel încât să aibă o dimensiune specificată.

Funcția are trei parametri: tabloul care trebuie completat, dimensiunea pe care trebuie să o aibă tabloul completat și valoarea cu care este completat tabloul.

Tabloul va fi completat astfel încât dimensiunea sa să fie egală cu valoarea absolută a celui de-al doilea parametru. În cazul în care valoarea acestui parametru este pozitivă, tabloul va fi completat la dreapta, iar dacă aceasta este

negativă, tabloul va fi completat la stânga. Dacă valoarea absolută a acestui parametru este mai mică sau egală cu dimensiunea tabloului transmis ca parametru, atunci nu sunt adăugate elemente. Funcția returnează tabloul obținut după completare.

Să considerăm vectorul definit prin:

```
$v = array(2,4,5,7);
```

În continuare vom prezenta câteva apeluri ale funcției și vectorii returnați de aceasta:

- `$r = array_pad($v2,6,-1): (2,4,5,7,-1,-1);`
- `$r = array_pad($v2,-7,0): (0,0,0,2,4,5,7);`
- `$r = array_pad($v2,3,5): (2,4,5,7);`

[*nu este adăugat nici un element*]

Funcțiile array_pop și array_push

Aceste funcții permit utilizarea tablourilor similar stivelor. Funcția `array_pop()` este utilizată pentru a elimina un element al tabloului (ultimul), iar funcția `array_push()` pentru a adăuga în tablou unul sau mai multe elemente.

Să considerăm că vectorul `$v` este vid. În urma execuției apelului `array_push($v,1,2,3)` acesta va conține elementele `1`, `2` și `3`. În acest moment, un apel de forma `array_pop($v)` va returna valoarea `3`, iar vectorul `$v` va conține elementele `1` și `2`.

În cazul în care vectorul transmis ca parametru este vid, funcția `array_pop()` returnează valoarea **`null`**. Funcția `array_push()` va returna întotdeauna noua dimensiune a vectorului.

Funcțiile array_shift și array_unshift

Perechile funcțiilor `array_pop()` și `array_push()` sunt `array_shift()` și `array_unshift()`. Acestea pot fi utilizate pentru a insera sau elimina elemente la începutul unui tablou. Executarea acestor funcții va duce la modificarea cheilor întregi astfel încât acestea să înceapă de la 0.

Funcția array_rand

Această funcție permite alegerea aleatoare a unui număr de elemente dintr-un vector.

Funcția are ca parametri un vector și o valoare numerică ce indică numărul elementelor care vor fi alese. Acest al doilea parametru poate lipsi, caz în care are valoarea implicită `1`.

În cazul în care se alege un singur element, funcția va returna cheia acestuia. În caz contrar, funcția va returna un tablou care va conține cheile elementelor alese.

În continuare vom prezenta un exemplu care ilustrează modul de utilizare a acestei funcții. Efectul este prezentat în figura 5.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția array_rand
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $v =
       array(2,4,5,7);
      $a =
       array_rand($v);
      echo "A fost
            ales
            elementul
            a cărui
            cheie este
            <B>$a</B>. Valoarea acestui
            element este
            <B>$v[$a]</B>.<BR><BR>";
      $a = array_rand($v);
      echo "A fost ales elementul a cărui
            cheie este <B>$a</B>. Valoarea
            acestui element este
            <B>$v[$a]</B>.<BR><BR>";
      $a = array_rand($v,2);
      echo "Au fost alese elementele ale
            căror chei sunt <B>$a[0]</B> și
            <B>$a[1]</B>. Valorile acestor
            elemente sunt <B>".$v[$a[0]].
            "</B> și <B>".$v[$a[1]].
            "</B>.<BR><BR>";
      $a = array_rand($v,2);
      echo "Au fost alese elementele ale
            căror chei sunt <B>$a[0]</B> și
            <B>$a[1]</B>. Valorile acestor
            elemente sunt <B>".$v[$a[0]].
            "</B> și <B>".$v[$a[1]]."</B>.";
    ?>
  </BODY>
</HTML>
```

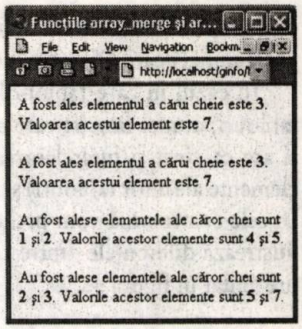


Figura 5: Funcția array_rand

Funcția array_reduce

Această funcție aplică o altă funcție transmisă ca parametru asupra elementelor unui tablou până în momentul în care tabloul este redus la o singură valoare.

Mai exact, funcția transmisă ca parametru este apelată pentru primele două elemente ale tabloului. Cele două elemente sunt eliminate și în tablou este inserat pe prima poziție rezultatul apelului funcției. Procedeul continuă până în momentul în care vectorul conține o singură valoare care este retunată.

Există și posibilitatea de a transmite o valoare inițială pentru efectuarea operațiilor. În acest caz funcția transmisă ca parametru va fi apelată pentru valoarea inițială și primul element al tabloului. În cazul în care tabloul este vid, rezultatul apelului va fi această valoare inițială.

Să considerăm funcțiile definite astfel:

```
function S($v,$w){
  return $v + $w;
}
```

```
function P($v,$w){
  return $v * $w;
}
```

În urma executării instrucțiunilor:

```
$a = array(2,4,5,7);
$x = array();
$b = array_reduce($a,"S");
$c = array_reduce($a,"P",10);
$d = array_reduce($x,"S",1);
```

valorile variabilelor `$b`, `$c` și `$d` vor fi `18` (`2+4+5+7`), `2800` (`10*2*4*5*7`), respectiv `1` (datorită faptului că vectorul `$x` este vid).

Funcția array_reverse

Această funcție inversează ordinea elementelor unui vector transmis ca parametru. În cazul în care este prezent un al doilea parametru și valoarea acestuia este **TRUE** (eventual după o conversie), atunci se păstrează corespondența dintre chei și valori. Funcția returnează tabloul obținut după inversare.

Să considerăm tabloul definit astfel:

```
$v = array("Gazeta","de","Informatică",
                    array("mai",2003));
```

În urma apelurilor `$a = array_reverse($v)` și `$b = array_reverse($v,TRUE)` tablourile `$a` și `$b` vor avea forma prezentată în figura 6.

```
Array                          Array
(                              (
    [0] => Array                   [3] => Array
        (                              (
            [0] => mai                     [0] => mai
            [1] => 2003                    [1] => 2003
        )                              )
    [1] => Informatică             [2] => Informatică
    [2] => de                      [1] => de
    [3] => Gazeta                  [0] => Gazeta
)                              )
```

Tabloul $a **Tabloul $b**

Figura 6: Efectul funcției array_reverse

În figură se observă clar diferențele dintre cheile celor două tablouri.

Funcția array_search

Această funcție verifică dacă un vector conține un element care are o anumită valoare și, dacă un astfel de element există, returnează cheia acestuia. În caz contrar funcția returnează valoarea **FALSE**. Primul parametru al funcției este valoarea căutată, iar al doilea este vectorul în care este căutată valoarea respectivă. Funcția permite folosirea unui al treilea parametru care este opțional. În cazul în care acesta există și are valoarea **TRUE**, atunci în timpul căutării se verifică și egalitatea tipurilor.

Funcția array_slice

Această funcție poate fi utilizată pentru a prelua o porțiune a unui tablou. Primul parametru reprezintă tabloul, al doilea reprezintă poziția la care începe porțiunea preluată, iar al treilea indică numărul elementelor preluate.

Această funcție ignoră cheile elementelor și ia în considerare pozițiile "reale" ale elementelor în tablou.

În cazul în care valoarea celui de-al doilea parametru este pozitivă, poziția de început este determinată relativ față de începutul tabloului. Pentru o valoare negativă, poziția este determinată relativ față de sfârșitul tabloului.

În cazul în care valoarea celui de-al treilea parametru este pozitivă, ea va indica numărul elementelor preluate. Dacă ea este negativă, atunci valoarea sa absolută va indica poziția față de sfârșitul vectorului la care se va "opri" preluarea elementelor. Acest parametru poate lipsi, caz în care sunt preluate toate elementele, până la sfârșitul tabloului.

Funcția va returna un tablou care va conține elementele preluate.

Vom prezenta în continuare modul în care poate fi utilizată această funcție pentru a prelua porțiuni ale unui vector. Efectul este ilustrat în figura 7.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția
          array_slice
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      function DisplayArray($v){
        echo "<B>";
        foreach ($v as $val)
          echo " $val";
        echo "</B><BR>";
      }
      $v = array("a","b","c","d","e");
      echo "<TT>Vectorul inițial:";
      DisplayArray($v);
      echo "array_slice(\$v,2):";
      DisplayArray(array_slice($v,2));
      echo "array_slice(\$v,2,-1):";
      DisplayArray(array_slice($v,2,-1));
      echo "array_slice(\$v,-2,1):";
      DisplayArray(array_slice($v,-2,1));
      echo "array_slice(\$v,0,3):";
      DisplayArray(array_slice($v,0,3));
    echo "</TT>";
    ?>
  </BODY>
</HTML>
```

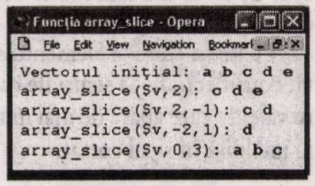


Figura 7: Funcția array_slice

Funcția array_splice

Efectul acestei funcții este similar efectului funcției `array_slice()`, diferența constând în faptul că porțiunea preluată este eliminată din tablou și înlocuită cu elementele unui al patrulea parametru care reprezintă un vector.

Funcția array_sum

Această funcție calculează suma elementelor unui tablou. În funcție de tipul elementelor, rezultatul va fi un număr

întreg sau un număr real. Pentru valorile nenumerice se încearcă efectuare unei conversii la numere întregi sau reale și în cazul în care valorile pot fi convertite (rezultatul conversiei nu este 0) ele sunt luate în considerare în momentul calculării sumei. De exemplu, în urma executării secvenței:

```
$a = array(2,4,6,8);
echo "suma(a) = ".array_sum($a)."\n";
$b = array("a"=>1.2,"b"=>2.3,"c"=>3.4);
echo "sum(b) = ".array_sum($b)."\n";
```

se va afișa:

```
sum(a) = 20
sum(b) = 6.9
```

Funcția array_unique

Această funcție elimină dintr-un vector toate valorile care apar de două sau mai multe ori și returnează vectorul obținut în urma eliminării.

Pentru a elimina elementele, funcția le sortează mai întâi considerându-le șiruri de caractere. Apoi, pentru fiecare valoare care apare de mai multe ori, va păstra prima cheie întâlnită în șirul sortat.

Funcții de sortare

Interpretorul *PHP* pune la dispoziție o mulțime de funcții care pot fi utilizate pentru a sorta elementele unui tablou. Acestea sunt (în ordine alfabetică):

- `arsort()` - sortează un tablou în ordine descrescătoare și păstrează asocierile dintre indici;
- `asort()` - sortează un tablou în ordine crescătoare și păstrează asocierile dintre indici;
- `krsort()` - sortează un tablou în ordine descrescătoare în funcție de cheile elementelor acestuia;
- `ksort()` - sortează un tablou în ordine crescătoare în funcție de cheile elementelor acestuia;
- `natcasesort()` - sortează un tablou folosind o "ordine naturală" fără a face distincție între literele mari și literele mici;
- `natsort()` - sortează un tablou folosind o "ordine naturală" făcând distincție între literele mari și literele mici
- `rsort()` - sortează un tablou în ordine descrescătoare în funcție de valorile elementelor acestuia;
- `sort()` - sortează un tablou în ordine crescătoare în funcție de valorile elementelor acestuia;
- `uasort()` - sortează un tablou folosind pentru comparări o funcție indicată de utilizator și păstrează asocierile dintre indici;
- `uksort()` - sortează un tablou în funcție de cheile acestuia folosind pentru comparări o funcție indicată de utilizator;
- `usort()` - sortează un tablou folosind pentru comparări o funcție indicată de utilizator.

Toate funcțiile necesită un parametru care reprezintă vectorul care trebuie sortat.

Funcțiile `arsort()`, `asort()`, `krsort()`, `ksort()`, `rsort()` și `sort()` permit utilizarea unui al doilea parametru (opțional) care este un indicator de tip. Valoarea implicită a acestuia este `SORT_REGULAR`. Al doilea parametru al funcțiilor `uasort()`, `uksort()` și `usort()` reprezintă identificatorul funcției care va fi folosită pentru comparare.

Funcțiile `natsort()` și `natcasesort()` nu permit utilizarea unui alt parametru și stabilesc ordinea așa cum ar face o ființă umană (ordine naturală).

De exemplu, pentru șirurile de caractere `IMG1.GIF`, `IMG2.GIF`, `IMG10.GIF` și `IMG12.GIF`, un algoritm "clasic" (bazat pe ordinea lexicografică) ar stabili ordinea: `IMG1.GIF`, `IMG10.GIF`, `IMG12.GIF` și `IMG2.GIF`, dar algoritmul care folosește ordinea naturală nu ar efectua nici o modificare.

Funcția compact

Această funcție are ca argumente un număr variabil de șiruri de caractere sau tablouri. Ea creează un tablou în care va insera toate șirurile care reprezintă o variabilă (aceste șiruri vor fi chei, iar valorile elementelor corespunzătoare vor fi valorile variabilelor). În cazul tablourilor, vor fi luate în considerare toate elementele care sunt șiruri de caractere, iar dacă tablourile conțin elemente care, la rândul lor, sunt tablouri, atunci procedeul se aplică recursiv și asupra acestora.

Funcțiile count și sizeof

Aceaste funcții returnează numărul elementelor unui vector transmis ca parametru. Teoretic, parametrul poate să nu fie un vector, dar în acest caz se va returna valoarea `1` (doar tablourile pot conține mai multe elemente). Nu există nici o diferență între aceste funcții.

Funcții pentru elementul curent

Fiecărui vector îi este asociată o referință care indică elementul curent. Există mai multe funcții care pot utiliza această referință. Acestea sunt (în ordine alfabetică):

- `current()` - returnează valoarea elementului curent al tabloului;
- `each()` - returnează cheia și valoarea elementului curent și deplasează referința la elementul următor; funcția returnează un tablou cu patru elemente: două dintre ele au cheile `0` și `key` și conțin cheia elementului; celelalte două au cheile `1` și `value` și conțin valoarea elementului;
- `end()` - deplasează referința la ultimul element al tabloului;
- `key()` - returnează cheia elementului curent al tabloului;
- `next()` - deplasează referința la următorul element al tabloului;
- `pos()` - returnează valoarea elementului curent al tabloului (are același efect ca și funcția `current()`);
- `prev()` - deplasează referința la elementul anterior al tabloului;
- `reset()` - deplasează referința la primul element al tabloului.

Funcția extract

Această funcție este utilizată pentru a crea variabile pe baza valorilor elementelor unui vector. Cheile elementelor vor deveni identificatori de variabile, iar valorile elementelor vor deveni valorile variabilelor respective.

Funcția verifică dacă cheile reprezintă identificatori valizi ai variabilelor sau dacă se încearcă crearea unei variabile care există deja. Aceste situații sunt tratate în funcție de valoarea unui al doilea parametru care reprezintă un indicator și este opțional. Valorile posibile pentru acest parametru sunt următoarele:

- `EXTR_OVERWRITE`: dacă variabila există deja, valoarea curentă va fi suprascrisă;
- `EXTR_OVERWRITE`: dacă variabila există deja, valoarea curentă nu va fi modificată;
- `EXTR_PREFIX_SAME`: dacă variabila există deja, denumirea noii variabile va fi prefixată cu un prefix transmis ca al treilea parametru al funcției;
- `EXTR_PREFIX_ALL`: denumirile tuturor variabilelor vor fi prefixate;
- `EXTR_PREFIX_INVALID`: în cazul în care o cheie nu reprezintă un identificator de variabilă validă, denumirea cheii va fi prefixată;
- `EXTR_IF_EXISTS`: nu sunt create variabile noi, ci sunt doar suprascrise valorile variabilelor care există deja;
- `EXTR_PREFIX_IF_EXISTS`: în cazul în care cheia reprezintă denumirea unei variabile, se creează o nouă variabilă prefixată; dacă nu există o astfel de variabilă, nu este creată o variabilă nouă;
- `EXTR_REFS`: variabilele create reprezintă referințe; așadar, prin intermediul lor pot fi modificate și elementele vectorului; acest indicator poate fi utilizat împreună cu oricare dintre celelalte cu ajutorul unei disjuncții logice.

Cel de-al treilea parametru al funcției trebuie să apară doar dacă valoarea celui de-al doilea este unul din indicatorii ai cărui identificator conține secvența `PREFIX`.

Al doilea parametru poate lipsi, caz în care are valoarea implicită `EXTR_OVERWRITE`.

Funcția in_array

Această funcție este similară funcției `array_search()`, singura diferență fiind faptul că, în cazul în care căutarea se încheie cu succes, nu este returnată cheia elementului găsit, ci valoarea `TRUE`.

Funcția range

Această funcție returnează un vector care conține toate valorile cuprinse între două limite date (`st` și `dr`). Primul element al vectorului va avea valoarea `st`, iar următoarele vor avea valorile `st + 1`, `st + 2` etc. dacă `st < dr`, respectiv `st - 1`, `st - 2` etc. dacă `st > dr`. Începând cu versiunea *PHP* 4.4.0 va exista și posibilitatea de a crea șiruri în care diferența dintre două elemente să fie diferită de `1`.

În cele ce urmează vom prezenta un document *PHP* care ilustrează modul de utilizare a acestei funcții. Efectul este ilustrat în figura 8.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția range
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      function DisplayArray($v){
        echo "<B>";
        foreach ($v as $val)
          echo " $val";
        echo "</B><BR>";
      }
      echo "<TT>";
      DisplayArray(range(1,10));
      DisplayArray(range(10,1));
      DisplayArray(range('a','k'));
      DisplayArray(range('k','a'));
      echo "</TT>";
    ?>
  </BODY>
</HTML>
```

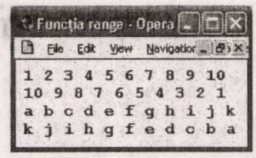


Figura 8: Funcția range

Funcția shuffle

Această funcție poate fi utilizată pentru a "amesteca" elementele unui tablou. Ea nu returnează nici o valoare, dar modifică ordinea elementelor tabloului.

Vom prezenta în continuare un document *PHP* care conține un exemplu de utilizare a acestei funcții, efectul fiind ilustrat în figura 9.

```
<HTML>
  <HEAD>
    <TITLE>
      Funcția shuffle
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      function DisplayArray($v){
        echo "<B>";
        foreach ($v as $val)
          echo " $val";
        echo "</B><BR>";
      }
      echo "<TT>Șirul inițial:<BR> ";
      $a = range(1,10);
      DisplayArray($a);
      echo "Șirul amestecat:<BR> ";
      shuffle($a);
      DisplayArray($a);
      echo "</TT>";
    ?>
  </BODY>
</HTML>
```

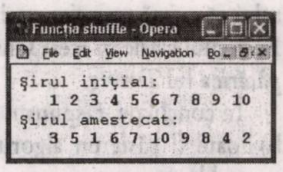


Figura 9: Funcția shuffle

BAC INFO' 2003

În cadrul acestui articol vom prezenta detalii referitoare la răspunsurile corecte la întrebările testului grilă prezentat în GInfo 13/4 (aprilie 2003). Pentru fiecare întrebare în parte vom preciza care dintre variantele de răspuns este cea corectă, precum și motivele pentru care aceasta a fost aleasă.

Întrebarea #1:
Există mai multe definiții ale conceptului de algoritm. O variantă care include, în principiu, toate caracteristicile unui algoritm este următoarea:

Un algoritm este o succesiune finită de pași care trebuie executați într-o anumită ordine și care duce la rezolvarea unei probleme.

Din această definiție rezultă imediat că afirmațiile corespunzătoare variantelor a) și c) sunt adevărate.

Afirmația corespunzătoare variantei d) este și ea adevărată deoarece, în practică, pot fi găsite mai multe modalități de rezolvare a unei probleme, aceste metode putând fi descrise cu ajutorul unor algoritmi diferiți.

Așadar, singura variantă rămasă este b). Afirmația corespunzătoare acesteia este într-adevăr falsă datorită faptului că există anumite probleme care nu pot fi rezolvate. Exemplul clasic în acest sens îl reprezintă *problema șchiopătării* a lui *Turing*.

În concluzie, răspunsul corect este:

b) Poate fi găsit un algoritm pentru rezolvarea oricărei probleme.

Întrebarea #2:
Afirmația corespunzătoare variantei a) este falsă, deoarece există expresii care pot să nu conțină variabile. Un exemplu în acest sens este adunarea a două constante (expresia `2 + 3` este validă și nu conține nici o variabilă).

Afirmația corespunzătoare variantei b) este adevărată deoarece, încă din definirea conceptului de variabilă, aceasta își poate schimba valoarea (de fapt, în marea majoritate a cazurilor, variabilele sunt utilizate pentru ca valorile lor să fie modificate pe parcursul executării algoritmilor).

Valoarea de adevăr a afirmației corespunzătoare variantei c) este discutabilă. Există anumite definiții ale expresiilor în care se precizează că acestea pot fi variabile, constante sau pot fi formate din doi operanzi (care sunt expresii) și un operator. Această afirmație ar putea fi interpretată astfel încât să rezulte că o expresie poate conține cel mult un operator. Totuși, în cazul în care expresiile folosesc ca operanzi alte expresii, acestea din urmă pot conține, la rândul lor, operatori. Așadar, o expresie "compusă" din mai multe expresii poate conține mai mulți operatori. De exemplu, expresia `1 + 2 + 3` conține doi operatori. În plus, deoarece afirmația de la punctul b) este categoric adevărată, este clar că nu putem considera și această afirmație ca fiind adevărată.

Afirmația de la punctul d) este falsă, deoarece există o mulțime de situații în care operațiile nu constau în efectuarea de calcule folosind variabile. Și în acest caz, exemplul efectuării adunării `1 + 2` este suficient pentru a arăta că afirmația este falsă.

În concluzie, răspunsul corect este:

b) Valoarea unei variabile se poate modifica pe parcursul executării unui algoritm.

Întrebarea #3:
În cazul unei structuri repetitive anterior condiționate, execuția corpului este precedată de testarea unei condiții. Dacă aceasta nu este îndeplinită, corpul nu se execută deloc.

Evident, este posibil ca această condiție să fie îndeplinită la a doua iterație. În acest caz, corpul instrucțiunii se va executa o singură dată.

De asemenea, este posibil ca această condiție să fie îndeplinită și la următoarele testări, deci este posibil să fie executat corpul instrucțiunii de mai multe ori.

În concluzie, răspunsul corect este:

d) de zero, una sau mai multe ori.

Întrebarea #4:
Cuvintele `for`, `while` și `if` sunt rezervate atât în limbajul *Pascal*, cât și în limbajul *C*. Ele nu pot fi folosite în alte scopuri. În cazul limbajului *Pascal*, `write` este identificatorul unei proceduri. Chiar dacă acest lucru nu este recomandat, el poate fi utilizat în alte scopuri. În cazul limbajului *C*, `printf` este identificatorul unei funcții și poate fi și el utilizat în alte scopuri deși acest lucru nu este recomandat.

În concluzie, răspunsul corect este:

Versiune Pascal: *Versiune C:*
c) `write` c) `printf`.

Întrebarea #5:
Potrivit sintaxei limbajului *Pascal*, pentru a defini un nou tip de date trebuie utilizat cuvântul cheie `type`. În cazul limbajului *C*, cuvântul cheie utilizat în acest scop este `typedef`.

În concluzie, răspunsul corect este:
Versiune Pascal: *Versiune C:*
a) `type` a) `typedef`.

Întrebarea #6:
Identificatorul folosit pentru a preciza tipul caracter este `char` atât în cazul limbajului *Pascal*, cât și în cazul limbajului *C*.

Pentru a atribui o valoare unei anumite variabile trebuie utilizat operatorul de atribuire care este `":="` pentru limbajul *Pascal* și `"="` pentru limbajul *C*.

În cazul ambelor limbaje, pentru a specifica faptul că se dorește folosirea unui anumit caracter, acest caracter trebuie cuprins între apostrofuri.

În concluzie, răspunsul corect este:
Versiune Pascal:
b) Declarație: `a:char;` Atribuire: `a:='H';`.
Versiune C:
b) Declarație: `char a;` Atribuire: `a='H';`.

Întrebarea #7:
Afirmația nu este adevărată deoarece pot exista funcții a căror listă de parametri să fie vidă.

În concluzie, răspunsul corect este:
b) Nu.

Întrebarea #8:
În cazul limbajului *Pascal*, pentru a determina codul *ASCII* al unui caracter trebuie utilizată funcția `ord`.

În cazul limbajului *C*, nu se face distincție între o variabilă de tip caracter și o variabilă de tip numeric. Caracterele sunt reprezentate direct prin codurile *ASCII*, deci pentru a determina un astfel de cod trebuie doar să utilizăm caracterul inclus între apostrofuri.

În concluzie, răspunsul corect este:
Versiune Pascal: *Versiune C:*
a) `a := ord('*');` a) `a = '*';`.

Întrebarea #9:
Un număr este impar dacă cea mai nesemnificativă cifră binară a sa este `1`. Rezultatul operației ȘI logic între o anumită valoare și valoarea `1` este ultima cifră a reprezentării binare deoarece toți biții reprezentării binare (cu excepția celui mai nesemnificativ) a valorii `1` sunt `0`. Celelalte operații logice pe biți nu pot fi utilizate pentru obținerea valorii celui mai nesemnificativ bit al reprezentării unui număr, motiv pentru care nu pot fi folosite pentru verificarea parității unui număr.

În concluzie, răspunsul corect este:
Versiune Pascal: *Versiune C:*
a) `a` **`and`** `1 = 1` c) `a & 1`.
Notă:
Acest tip de test poate fi utilizat numai dacă valoarea variabilei `a` este un număr întreg pozitiv.

Întrebarea #10:
Din definiția tabloului rezultă că acesta este o structură statică de date ale cărei elemente au același tip. Așadar, afirmațiile corespunzătoare punctelor a) și b) sunt adevărate.

Afirmația corespunzătoare punctului c) este și ea adevărată deoarece, pentru declararea unui tablou în limbajul *Pascal*, trebuie utilizat cuvântul cheie `array`, în timp ce declararea unui tablou în limbajul *C* se realizează fără a utiliza acest cuvânt cheie.

Afirmația corespunzătoare punctului d) este falsă în cazul limbajului *Pascal*, deoarece domeniul unui tablou poate fi orice tip ordinal (cu restricțiile impuse de dimensiunea segmentului de date), așadar primul element poate avea indicele diferit de `1`. Ea este falsă și în cazul limbajului *C* deoarece în cazul acestui limbaj de programare indicele primului element al unui tablou este `0`.

În concluzie, răspunsul corect este:
Versiune Pascal:
d) Primul element al unui tablou are întotdeauna indicele 1.
Versiune C:
d) Primul element al unui tablou are întotdeauna indicele 1.

Întrebarea #11:
În cazul limbajului *Pascal*, funcția care returnează numărul de caractere dintr-un string este `length`.

În cazul limbajului *C*, funcția care returnează această valoare este `strlen`.

În concluzie, răspunsul corect este:
Versiune Pascal: *Versiune C:*
c) `length` c) `strlen`.

Întrebarea #12:
În cazul limbajului *Pascal* un articol este declarat cu ajutorul cuvântului cheie **`record`**. Câmpurile trebuie să apară după acest cuvânt și sunt declarate prin identificatorul și tipul câmpului. Urmează cuvântul cheie **`end`** care indică terminarea secvenței de declarare a articolului.

În cazul limbajului *C* un articol este declarat cu ajutorul cuvântului cheie **`struct`**. Câmpurile trebuie să apară după acest cuvânt, sunt declarate prin identificatorul și tipul câmpului și sunt cuprinse între acolade.

În concluzie, răspunsul corect este:
Versiune Pascal:
a) **`var`** `a:`**`record`** `x,y:real` **`end`**`;`
Versiune C:
a) **`struct`** `{`**`float`** `x,y;}a;`.

Întrebarea #13:
Un fișier text poate conține și alte caractere în afara literelor alfabetului latin (de exemplu cifre, semne de punctuație

etc.), motiv pentru care afirmația corespunzătoare punctului a) este falsă.

Pentru a citi date dintr-un fișier text acesta trebuie deschis pentru citire; așadar afirmația corespunzătoare punctului b) este adevărată.

Fișierele text pot fi deschise pentru adăugare la sfârșitul lor, așadar pot fi adăugate litere. Ca urmare, afirmația corespunzătoare punctului c) este falsă.

Dimensiunea unui fișier text este limitată doar de capacitatea de stocare a discului hard și de sistemul de fișiere utilizat. În orice caz, aceasta nu este limitată la 65535 de caractere, motiv pentru care afirmația corespunzătoare punctului d) este falsă.

În concluzie, răspunsul corect este:

b) Înainte de a prelua date dintr-un fișier text, acesta trebuie deschis pentru citire.

Întrebarea #14:

Tipul corespunzător fișierelor text în limbajul *Pascal* este `text`. Acesta este un identificator de tip predeclarat, nefiind un cuvânt rezervat.

În cazul limbajului *C* există mai multe posibilități de declarare a unei variabile care să reprezinte un fișier. Dintre variantele disponibile, doar construcția `FILE*` poate fi utilizată pentru a declara o astfel de variabilă. Nici această construcție nu reprezintă un cuvânt rezervat.

În concluzie, răspunsul corect este:

Versiune Pascal:

d) nu există nici un astfel de cuvânt cheie.

Versiune C:

d) nu există nici un astfel de cuvânt cheie.

Întrebarea #15:

În cazul limbajului *Pascal*, pentru a realiza corespondența dintre o variabilă și un anumit fișier se utilizează procedura `assign`.

În cazul limbajului *C* există mai multe modalități de a realiza această corespondență. Dintre variantele prezentate, poate fi utilizată funcția `fopen`.

În concluzie, răspunsul corect este:

Versiune Pascal: a) `assign`

Versiune C: a) `fopen`.

Întrebarea #16:

Secvența prezentată reprezintă descrierea algoritmului clasic de determinare a inversului unui număr. Așadar, răspunsul corect este:

c) determină inversul unui număr (numărul obținut prin citirea cifrelor de la dreapta la stânga).

Întrebarea #17:

Datorită condiției `d < rad` se vor testa divizorii primi strict mai mici decât rădăcina pătrată a numărului a cărui primalitate este verificată. Așadar, dacă un număr este multiplu al rădăcinii sale pătrate (și nu este multiplu al nici unui alt număr mai mic, diferit de 1), atunci numărul va fi considerat a fi prim, cu toate că are un divizor diferit de `1` și de el însuși. Ca urmare, algoritmul nu va funcționa corect în cazul în care se verifică primalitatea unor numere care sunt pătrate perfecte ale unor numere prime. Dintre cele trei numere, singurul care respectă această proprietate este `25`.

Algoritmul nu va intra în ciclu infinit deoarece valoarea numărului testat crește cu `2` la fiecare pas. Până la urmă acesta va ajunge mai mare sau egal cu rădăcina pătrată a numărului a cărui primalitate este testată sau se va determina un divizor.

În concluzie, răspunsul corect este:

d) `25`.

Întrebarea #18:

Primele elemente ale șirului lui *Fibonacci* sunt: `0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597...`

Cel mai mic element format din trei cifre este 144 care este par, deci nu este prim. Următorul element, 233, este prim, deci este cel mai mic element prim al șirului lui *Fibonacci* care are trei cifre.

În concluzie, răspunsul corect este:

a) `233`.

Întrebarea #19:

Elementul neutru al unei operații este acea valoare pentru un operand a cărei utilizare, indiferent de valoarea celuilalt operand, să determine ca rezultatul să aibă aceeași valoare cu acest al doilea operand. Pentru minimul a două numere valoarea corespunzătoare este ∞, deoarece orice număr este mai mic decât ∞.

În concluzie, răspunsul corect este:

b) ∞.

Întrebarea #20:

Algoritmul de sortare prin inserție constă în parcurgerea elementelor vectorului și determinarea poziției elementului curent în șirul parțial sortat. Așadar, răspunsul corect este:

a) poziția elementului curent în șirul parțial sortat la pașii anteriori.

Întrebarea #21:

Pentru a putea interclasa două șiruri este obligatoriu ca acestea să fie sortate. Așadar, răspunsul corect este:

b) șirurile trebuie să fie ordonate în funcție de un același criteriu.

Întrebarea #22:

În cazul căutării binare, la fiecare pas, lungimea șirului în care se caută numerele se înjumătățește. Așadar, dacă șirul are N elemente, după $[\log_2 N] + 1$ pași, lungimea șirului va fi 1, deci căutarea se va încheia.

Datorită faptului că avem $N = 1000$ este evident că varianta corectă este:

b) cuprins între `7` și `13`.

Întrebarea #23:
Există două modalități de transmitere a parametrilor și anume: prin valoare și prin referință. Așadar, răspunsul corect este:
c) valoare sau referință.

Întrebarea #24:
Afirmația corespunzătoare punctului a) este falsă, deoarece numărul parametrilor unui subprogram nu este limitat la 12.

Afirmația corespunzătoare punctului b) este falsă, pentru că există posibilitatea de a scrie subprograme care nu conțin structuri repetitive.

Afirmația corespunzătoare punctului c) este falsă, deoarece un subprogram poate conține orice tip de instrucțiuni, inclusiv una prin care se apelează un alt subprogram.

În concluzie, răspunsul corect este:
d) nici una dintre cele trei.

Întrebarea #25:
În cazul limbajului *Pascal*, varianta corespunzătoare punctului a) este corectă deoarece se respectă sintaxa limbajului.

Varianta corespunzătoare punctului b) este incorectă pentru că tipul unui parametru nu poate fi un vector decât dacă se definește acel vector ca tip.

Varianta corespunzătoare punctului c) nu este corectă deoarece procedurile nu au tip.

Varianta corespunzătoare punctului d) nu este corectă deoarece identificatorul procedurii nu poate fi `procedure`, acesta fiind un cuvânt rezervat.

În cazul limbajului *C* situația este mult mai clară, în acest limbaj neexistând proceduri.

În concluzie, răspunsul corect este:
Versiune Pascal:
a) `procedure proc(var a:byte);`.
Versiune C:
a) în limbajul *C* nu există proceduri.

Întrebarea #26:
Datorită faptului că variabila `a` este folosită și ca variabilă locală în cadrul subprogramului `aduna`, modificarea acestei valori în interiorul suprogramului nu are efect în afara acestuia. Ca urmare, în urma apelului, valoarea variabilei globale `a` va rămâne `3`.

Variabila `b` nu este definită în interiorul subprogramului, deci în momentul în care valoarea ei se modifică, operația poate fi efectuată doar asupra variabilei globale. Ca urmare, variabila globală `b` va avea valoarea `5` în urma execuției apelului.

În concluzie, răspunsul corect este:

Versiune Pascal:	*Versiune C:*
c) `3 5`	c) `3 5`.

Întrebarea #27:
Există două posibiltăți de implementare a recursivității: prin autoapelul unui subprogram (recursivitate directă) și prin apelul unui subprogram care la rândul său, apelează primul subprogram (recursivitate indirectă).

În concluzie, răspunsul corect este:
a) directă sau indirectă.
Notă:

Recursivitatea indirectă poate fi realizată și cu ajutorul a trei sau mai multe subprograme care se apelează succesiv, iar ultimul îl apelează pe primul.

Întrebarea #28:
Datorită faptului că funcția determină un produs și în cazul în care valoarea parametrului *n* este `0`, rezultatul este `0`, rezultatul final va fi `0` deoarece se vor realiza doar înmulțiri cu `0`.

Nu se va intra în ciclu infinit deoarece valoarea parametrului scade la fiecare pas, deci va deveni `0` până la urmă.

În concluzie, răspunsul corect este:

Versiune Pascal:	*Versiune C:*
c) `0`	c) `0`.

Notă:

Dacă valoarea returnată ar fi fost `1` pentru cazul în care parametrul ar avea valoarea `0`, funcția ar fi determinat factorialul unui număr.

Întrebarea #29:
În cazul în care problema este definită recursiv nu este obligatoriu (de obicei nici necesar) să fie utilizată metoda *backtracking*. Folosirea ei este indicată într-o mulțime de situații în care trebuie determinate toate soluțiile unei probleme. De obicei, algoritmii care utilizează această metodă sunt foarte lenți, deci ea nu este recomandată în cazul în care trebuie găsit cel mai rapid algoritm (decât dacă nu există o alternativă care duce la obținerea unui algoritm eficient).

În concluzie, răspunsul corect este:
b) când trebuie determinate toate soluțiile unei probleme.

Întrebarea #30:
Algoritmul se oprește după executarea a `m` pași (la începutul celui de-al (`m + 1`) - lea pas), și la fiecare pas se încearcă adăugarea de elemente care sunt mai mari decât elementul de la pasul anterior. Așadar, elementele șirului care reprezintă o soluție sunt ordonate crescător.

Ca urmare, pentru o valoare `n`, se determină toate șirurile formate din `m` elemente distincte, ordonate crescător. Datorită acestei ordonări, nu există posibilitatea de a găsi două șiruri care să conțină aceleași elemente, dar în altă ordine. Deci, aceste șiruri pot fi considerate ca fiind mulțimi.

În concluzie, răspunsul corect este:
Versiune Pascal:
d) determină toate submulțimile mulțimii `{1, 2, ..., n}` care au `m` elemente.
Versiune C:
d) determină toate submulțimile mulțimii `{1, 2, ..., n}` care au `m` elemente.

Întrebarea #31:
Este evident că pe fiecare linie se poate așeza un singur turn. Așadar, va trebui să determinăm posibilitățile de a alege coloanele pe care se vor așeza cele opt turnuri. Datorită faptului că pe fiecare coloană poate apărea un singur turn și nu există restricții suplimentare, soluțiile vor fi date de toate posibilitățile de a alege o ordine în care sunt scrise numerele cuprinse între `1` și `8`. Așadar, problema este echivalentă cu determinarea tuturor permutărilor șirului (`1, 2, 3, 4, 5, 6, 7, 8`).

În concluzie, răspunsul corect este:
b) problema determinării tuturor permutărilor mulțimii `{1, 2, 3, 4, 5, 6, 7, 8}`.

Întrebarea #32:
Eroarea apare în linia `21` pentru versiunea *Pascal*, respectiv linia `17` pentru versiunea C, deoarece condiția `pas < n` permite executarea a cel mult `n - 1` pași. Așadar, o submulțime cu `n` elemente (deci submulțimea care conține toate elementele mulțimii) nu va putea fi determinată.

În concluzie, răspunsul corect este:

Versiune Pascal:	*Versiune C:*
c) `21`	c) `17`.

Întrebarea #33:
Se observă că subprogramul `Back` este apelat cu valoarea `n` pentru parametrul `pas` și, prin autoapelare, valoarea parametrului scade.

O eventuală soluție este afișată doar atunci când valoarea parametrului `pas` este `n + 1`, ceea ce nu este posibil datorită modului în care variază valoarea acestui parametru.

Așadar, subprogramul `ScrieSolutie` nu va fi apelat niciodată, deci pe ecran nu se va scrie nimic.

În concluzie, răspunsul corect este:

Versiune Pascal:	*Versiune C:*
a) nimic	a) nimic.

Întrebarea #34:
Numărul permutărilor unui șir cu N elemente este $N!$. Numărul submulțimilor unei mulțimi cu N elemente este 2^N. Numărul submulțimilor care au exact M elemente este C_N^M.

Submulțimile care au cel puțin 3 elemente, au evident, 3, 4, 5 sau 6 elemente. Numărul total al acestora va fi:

$$C_6^3 + C_6^4 + C_6^5 + C_6^6 = 20 + 15 + 6 + 1 = 42.$$

Pentru $N = 6$ vom avea $6! = 720$ permutări și $2^6 = 64$ submulțimi.

În concluzie, răspunsul corect este:
b) `a = 720, b = 64, c = 42`.

Întrebarea #35:
Afirmația corespunzătoare punctului a) este adevărată pentru că stiva funcționează pe baza principiului ultimul sosit, primul servit. Așadar, ultimele elemente inserate în stivă sunt eliminate primele. Ca urmare, elementele sunt eliminate în ordinea inversă inserării lor.

Afirmația corespunzătoare punctului b) este adevărată deoarece coada funcționează pe baza principiului primul sosit, primul servit. Așadar, primele elemente inserate în coadă sunt eliminate primele. Ca urmare, elementele sunt eliminate în ordinea inserării lor.

Afirmația corespunzătoare punctului c) este adevărată pentru că recursivitatea constă în copierea pe o stivă a diferitelor informații necesare unui apel a unui subprogram (parametri, adresă de retur) și eliminarea lor în momentul încheierii execuției apelului. Ca urmare, aceste operații pot fi simulate dacă se utilizează o altă stivă.

Afirmația corespunzătoare punctului d) este falsă deoarece recursivitatea (atât cea directă, cât și cea indirectă) poate fi simulată folosind o stivă și nu o coadă.

În concluzie, răspunsul corect este:
d) cozile pot fi utilizate pentru a simula recursivitatea indirectă.

Întrebarea #36:
Pentru a insera un element într-o listă dublu înlănțuită trebuie modificate referințele sale spre predecesorul și succesorul său. Succesorul predecesorului va fi noul element, iar predecesorul succesorului va fi tot noul element, deci trebuie modificate încă două referințe.

Așadar, se modifică un număr total de patru referințe, deci răspunsul corect este:
b) patru referințe.

Întrebarea #37:
În urma execuției operațiilor `Adaugă_în_stivă(1);` și `Adaugă_în_stivă(2);` stiva va conține elementele `1` și `2`, vârful stivei fiind `2`. Acesta este eliminat în urma execuției operației `Elimină_primul_element_al_stivei;` deci stiva va conține un singur element: `1`.

În continuare se adaugă elementele `3` și `4` (prin operațiile `Adaugă_în_stivă(3);` și `Adaugă_în_stivă(4);`), stiva conținând elementele `1`, `3` și `4`, iar vârful fiind `4`. Acesta va fi scris de două ori deoarece se execută de două ori operația `Scrie_primul_element_al_stivei;`. Pe ecran va apărea (în acest moment) **4 4**.

Acum, prin intermediul operațiilor `Adaugă_în_stivă(5);` și `Adaugă_în_stivă(6);`, vom adăuga în stivă elementele `5` și `6`, aceasta conținând acum cinci elemente: `1`, `3`, `4`, `5` și `6`, vârful fiind `6`.

Prin execuția operației `Elimină_primul_element_al_stivei;` acest vârf este eliminat, stiva va rămâne cu patru elemente: `1`, `3`, `4` și `5` , iar noul vârf va fi `5`.

Acest nou vârf este și el eliminat în urma execuției următoarei operații `Elimină_primul_element_al_stivei;` deci vom avea trei elemente în stivă (`1`, `3` și `4`) iar vârful va fi `4`.

Operațiile `Adaugă_în_stivă(7);` și `Adaugă_în_stivă(8);` vor duce la adăugarea elementelor `7` și `8`; stiva va conține elementele `1`, `3`, `4`, `7` și `8`, vârful fiind `8`. Acesta este

afișat în urma execuției operației `Scrie_primul_element_al_stivei;`, deci pe ecran va apărea acum **4 4 8**.

Prin execuția operației `Adaugă_în_stivă(9);` se adaugă elementul 9 care devine noul vârf al stivei care conține acum elementele 1, 3, 4, 7, 8 și 9.

Noul vârf este imediat eliminat datorită operației `Elimină_primul_element_al_stivei;` deci revenim în situația în care avem în stivă elementele 1, 3, 4, 7 și 8, iar vârful este 8.

Prin operația `Scrie_primul_element_al_stivei;` se va afișa valoarea acestui vârf (8), deci pe ecran vom avea **4 4 8 8**.

Următoarea operație (`Adaugă_în_stivă(10);`) duce la adăugarea elementului 10. Vom avea o stivă cu șase elemente (1, 3, 4, 7, 8 și 10) al cărei vârf este 10.

Operația `Elimină_primul_element_al_stivei;` va duce la eliminarea elementului 10 din vârful stivei, deci, din nou, aceasta va conține elementele 1, 3, 4, 7 și 8, vârful fiind 8.

Următoarea operație (`Elimină_primul_element_al_stivei;`) va duce la eliminarea vârfului (8), deci vom rămâne cu o stivă care conține patru elemente (1, 3, 4 și 7) care are în vârf elementul 7.

Această valoare va fi afișată datorită execuției operației `Scrie_primul_element_al_stivei;`. În acest moment, pe ecran, vom avea **4 4 8 8 7**.

Operația `Elimină_primul_element_al_stivei;` va duce la eliminarea elementului 7 din vârful stivei. Acum, aceasta va conține elementele 1, 3 și 4, vârful fiind 4.

Prin următoarea operație (`Elimină_primul_element_al_stivei;`) se va elimina și elementul 4, deci vom avea o stivă cu două elemente (1 și 3) care are vârful 3.

Valoarea este scrisă datorită operației `Scrie_primul_element_al_stivei;` deci pe ecran va apărea acum **4 4 8 8 7 3**.

Vom rămâne cu un singur element (1) după execuția operației `Elimină_primul_element_al_stivei;` care va fi afișat datorită operației `Scrie_primul_element_al_stivei;`. În acest moment pe ecran vom avea **4 4 8 8 7 3 1**.

Acest ultim element este eliminat datorită execuției instrucțiunii (`Elimină_primul_element_al_stivei;`), deci stiva va rămâne vidă în urma executării tuturor operațiilor.

În concluzie, suma elementelor afișate este 4 + 4 + 8 + 8 + 7 + 3 + 1 = 35, deci răspunsul corect este:

a) 35.

Întrebarea #38:

Potrivit definiției conceptului de santinelă, acesta este un element fictiv care este utilizat pentru a ușura modul în care se realizează anumite operații cu structurile dinamice de date el nefăcând parte din structură din punct de vedere teoretic. Ca urmare, răspunsul corect este:

a) un element fictiv care nu face parte din structura propriu-zisă și care este utilizat pentru simplificarea implementării operațiilor de inserare și eliminare.

Întrebarea #39:

Afirmația corespunzătoare punctului a) este falsă datorită faptului că din definiția grafurilor neorientate nu rezultă obligativitatea existenței unei muchii. De exemplu, un graf poate fi format dintr-un singur vârf.

Afirmația corespunzătoare punctului b) este adevărată deoarece definiția grafurilor neorientate nu impune nici o limită asupra numărului nodurilor izolate.

Afirmația corespunzătoare punctului c) este adevărată deoarece o muchie leagă două vârfuri, deci este identificată în mod unic prin perechea de noduri care reprezintă extremitățile ei.

Afirmația corespunzătoare punctului d) este adevărată deoarece respectă definiția lungimii unui ciclu.

În concluzie, răspunsul corect este:

a) un graf neorientat trebuie să conțină cel puțin o muchie.

Întrebarea #40:

Afirmația corespunzătoare punctului a) este falsă pentru că un subgraf se obține prin eliminarea unor noduri ale unui graf, deci numărul acestora va fi mai mic în subgraf decât în graful inițial.

Afirmația corespunzătoare punctului b) este falsă deoarece conexitatea unui graf nu este o condiție suficientă (deși este necesară) pentru ca în graful respectiv să existe un ciclu eulerian.

Afirmația corespunzătoare punctului c) este adevărată deoarece, într-un graf complet cu N noduri, fiecare vârf este legat prin câte o muchie de toate celelalte N - 1 noduri. Așadar, toate nodurile au gradul N - 1, deci un graf complet este întotdeauna și un graf regulat.

Afirmația corespunzătoare punctului d) este falsă deoarece există o mulțime de grafuri hamiltoniene care nu sunt și grafuri regulate.

În concluzie, răspunsul corect este:

c) un graf complet este întotdeauna și un graf regulat.

Întrebarea #41:

O linie (sau o coloană) a matricei de adiacență va conține un număr de elemente cu valoarea 1 egal cu numărul muchiilor care au ca extremitate nodul corespunzător liniei respective. Așadar, gradul unui nod este dat de numărul elementelor cu valoarea 1 de pe linia (coloana) corespunzătoare.

Ca urmare, cele 10 noduri ale grafului vor avea gradele 5, 8, 4, 5, 6, 4, 4, 4, 4, respectiv 4.

În concluzie, cel mai mare grad este 8, deci răspunsul corect este:

c) 8.

Întrebarea #42:

Într-un graf neorientat gradul unui vârf este dat de numărul total al arcelor care "intră" sau "ies" din nodul respectiv. Numărul arcelor care "intră" într-un nod reprezintă gradul interior, iar numărul celor care "ies" reprezintă gradul exterior.

Așadar, gradul unui nod va fi suma dintre gradul interior și cel exterior, deci răspunsul corect este:
c) suma dintre gradul interior și gradul exterior.

Întrebarea #43:
Variantele de la punctele a), b) și c) sunt corecte, ele descriind exact corespondențele dintre noțiuni (muchie - arc, ciclu - circuit, lanț - drum).

Varianta de la punctul d) nu este corectă deoarece noțiunea de adiacență este aceeași atât pentru grafurile orientate, cât și pentru cele neorientate. Noțiunea de incidență are o altă semnificație.

În concluzie, răspunsul corect este:
d) noțiunii de adiacență în grafurile neorientate îi corespunde noțiunea de incidență în grafurile orientate.

Întrebarea #44:
Afirmația corespunzătoare punctului a) este adevărată pentru că o frunză este legată printr-o muchie doar de părintele său, deci are gradul 1.

Afirmația corespunzătoare punctului b) este adevărată dat fiind faptul că toate nodurile (cu excepția rădăcinii) sunt fii ai rădăcinii, ai fiilor rădăcinii, ai fiilor fiilor rădăcinii etc., deci au un părinte.

Afirmația corespunzătoare punctului c) este falsă deoarece într-un arbore un nod poate avea mai mulți fii (care sunt toți frați). Un nod poate avea cel mult un frate doar în arborii binari.

În concluzie, răspunsul corect este:
c) orice nod al unui arbore cu rădăcină poate avea cel mult un frate.

Întrebarea #45:
Datorită faptului că există muchiile (1, 2) și (1, 5), nodurile 2 și 5 vor avea ca părinte nodul 1.

Singurul nod care este legat de nodul 2 printr-o muchie și pentru care nu am stabilit încă părinții este 7 (există o muchie între 2 și 7). Așadar, părintele nodului 7 va fi 2.

De nodul 5 sunt legate prin muchiile (4, 5) și (3, 5) nodurile 3 și 4, deci acestea îl vor avea ca părinte pe 5.

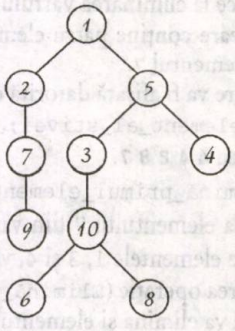
Nodul 9 este legat de nodul 7 prin muchia (7, 9), deci 7 va fi părintele nodului 9.

Nodul 10 este legat de nodul 3 prin muchia (3, 10), deci părintele nodului 10 va fi 3.

În sfârșit, de nodul 10 sunt legate prin muchiile (6, 10) și (8, 10) nodurile 6 și 8 care îl vor avea ca părinte.

Rezultă că:
- părintele nodului 2 este 1;
- părintele nodului 3 este 5;
- părintele nodului 4 este 5;
- părintele nodului 5 este 1;
- părintele nodului 6 este 10;
- părintele nodului 7 este 2;
- părintele nodului 8 este 10;
- părintele nodului 9 este 7;
- părintele nodului 10 este 3.

Arborele cu rădăcină definit prin lista de muchii dată este reprezentat grafic în figura următoare:



În concluzie, răspunsul corect este:
a) 1 5 5 1 10 2 10 7 3.

Despre testul GInfo

Deși a avut doar un caracter orientativ, testul grilă publicat în *GInfo* a atras atât atenția cititorilor, cât și a multor elevi care nici măcar nu au auzit până acum de existența revistei noastre.

Versiunea *online* a testului a fost folosită de mii de elevi care au dorit să își evalueze cunoștințele.

Testul a fost bine primit și de profesori, deoarece în momentul publicării lui nu exista un alt test care să poată fi utilizat pentru familiarizarea elevilor cu modul în care se va desfășura în acest an examenul de *Bacalaureat* la disciplina *Informatică*.

Între timp, au apărut și teste furnizate de *Ministerul Educației și Cercetării*. Deși ar putea părea bizar, și caracterul acestora este doar orientativ. De exemplu, metoda *backtracking* lipsește din aceste teste chiar dacă, credem noi, ea va apărea cu siguranță la examen.

Încheiem prin a mulțumi tuturor celor care s-au arătat interesați de testul nostru, mulțumim tuturor celor care ne-au felicitat și urăm mult succes tuturor elevilor care se pregătesc în vederea susținerii examenului de Bacalaureat la începutul acestei veri.

Grila corectă

1. a.☐ b.☒ c.☐ d.☐	16. a.☐ b.☐ c.☒ d.☐	31. a.☐ b.☒ c.☐ d.☐
2. a.☐ b.☒ c.☐ d.☐	17. a.☐ b.☐ c.☐ d.☒	32. a.☐ b.☐ c.☒ d.☐
3. a.☐ b.☐ c.☐ d.☒	18. a.☒ b.☐ c.☐ d.☐	33. a.☒ b.☐ c.☐ d.☐
4. a.☐ b.☐ c.☒ d.☐	19. a.☐ b.☒ c.☐ d.☐	34. a.☐ b.☒ c.☐ d.☐
5. a.☒ b.☐ c.☐ d.☐	20. a.☒ b.☐ c.☐ d.☐	35. a.☐ b.☐ c.☐ d.☒
6. a.☐ b.☒ c.☐ d.☐	21. a.☐ b.☒ c.☐ d.☐	36. a.☐ b.☒ c.☐ d.☐
7. a.☐ b.☒ c.☐ d.☐	22. a.☐ b.☒ c.☐ d.☐	37. a.☒ b.☐ c.☐ d.☐
8. a.☒ b.☐ c.☐ d.☐	23. a.☐ b.☐ c.☒ d.☐	38. a.☒ b.☐ c.☐ d.☐
9. a.☐ b.☐ c.☒ d.☐	24. a.☐ b.☐ c.☐ d.☒	39. a.☒ b.☐ c.☐ d.☐
10. a.☐ b.☐ c.☐ d.☒	25. a.☒ b.☐ c.☐ d.☐	40. a.☐ b.☐ c.☒ d.☐
11. a.☐ b.☐ c.☒ d.☐	26. a.☐ b.☐ c.☒ d.☐	41. a.☐ b.☐ c.☒ d.☐
12. a.☒ b.☐ c.☐ d.☐	27. a.☒ b.☐ c.☐ d.☐	42. a.☐ b.☐ c.☒ d.☐
13. a.☐ b.☒ c.☐ d.☐	28. a.☐ b.☐ c.☒ d.☐	43. a.☐ b.☐ c.☐ d.☒
14. a.☐ b.☐ c.☐ d.☒	29. a.☐ b.☒ c.☐ d.☐	44. a.☐ b.☐ c.☒ d.☐
15. a.☒ b.☐ c.☐ d.☐	30. a.☐ b.☐ c.☐ d.☒	45. a.☒ b.☐ c.☐ d.☐

COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui ultim episod al serialului dedicat compresiilor de date vă vom prezenta modul de prelucrare și comprimare a sunetelor și imaginilor.

Dacă în numerele anterioare ale revistei am prezentat tehnici de compresie a datelor fără pierderi de informație, în cele ce urmează vă vom prezenta câteva tehnici de compresie cu pierdere de date. Compresia de date cu pierdere de informație se aplică în special pentru date multimedia cum ar fi sunete, imagini și secvențe video.

Prelucrarea sunetelor

Există multe persoane care și-au pus întrebarea "*Cum sunt stocate sunetele în cadrul unui fișier?*". În continuare vă vom prezenta transformările prin care trece un sunet de la generarea lui de către un dispozitiv sau o persoană până în momentul scrierii lui într-un fișier.

Captarea și stocarea sunetelor

Sunetele generate de dispozitive sau persoane sunt captate de dispozitive de captare a sunetelor.

Cel mai bun exemplu de dispozitiv de captare a sunetelor este microfonul. Principiul de funcționare al acestuia este similar modului de funcționare a urechii umane, prin urmare, sunetele care ajung la microfon interacționează cu membrana acestuia care generează semnale electrice care reprezintă amplitudinea sunetului în funcție de timp.

În continuare, semnalele electrice generate de microfon trebuie transformate în șiruri de numere. Acest lucru se realizează de către o unitate *ADC* (*Analog Digital Convertor*) care primește la intrare o tensiune electrică și generează la ieșire un număr reprezentat binar. Datorită faptului că orice tensiune electrică se reprezintă numeric printr-un șir finit de biți, apare o eroare numită *eroare de cuantificare*.

Pentru reprezentarea unui semnal în formă digitală, acesta trebuie eșantionat. Prin eșationare, dintr-un semnal se extrag cadre la intervale de ΔT secunde.

După eșantionarea unui semnal, acesta poate fi folosit de aplicații multimedia pentru a prelucra și/sau stoca șirurile de biți generate de unitățile *ADC*.

Calitatea sunetului digital care se dorește a fi obținută depinde de numărul de biți folosiți pentru a reprezenta un cadru și de intervalul de timp dintre cadre. Calitatea sunetului este mai bună cu cât numărul de biți este mai mare și cu cât intervalul de timp dintre două cadre este mai mic.

De exemplu, pentru sistemul telefonic, cadrele sunt reprezentate folosindu-se 7 sau 8 biți, intervalul dintre cadre este 1/8000 secunde (8000 cadre/sec) și se pierd toate sunetele cu frecvențe mai mari de 4 kHz.

Un alt exemplu îl constituie *CD*-urile audio. Pe un *CD*-audio sunetele sunt înregistrate folosindu-se 16 biți pentru a reprezenta un cadru și intervalul dintre cadre este 1/44.100 (44.100 cadre/sec).

Compresia sunetelor digitale

Datorită faptului că pentru a transmite la distanță sunete cu o calitate foarte bună (de calitatea *CD*-urilor audio) sunt necesari 1,411 Mbps pentru semnale audio stereo și 705,6 Kbps pentru semnale audio mono. Aceste numere sunt foarte mari și s-a pus problema compresiei sunetelor.

Cei mai importanți algoritmi utilizați în procesul de compresie a sunetelor sunt ***codificarea formei de undă*** și ***codificarea perceptivă***.

Prin *codificarea formei de undă*, semnalul este transformat cu ajutorul transformatei *Fourier* în componentele sale în frecvență (reprezentarea amplitudinlor în funcție de timp). După aplicarea transformatei, amplitudinea fiecărei componente este codificată minimal, folosind cât mai puțini biți. Transformata *Fourier* duce la pierdere de informație.

Șirul de biți obținuți se comprimă folosind unul dintre algoritmii prezentați în episoadele anterioare ale compresiilor de date. Scopul este de a reproduce exact forma de undă, la celălalt capăt folosind cât mai puțini biți.

Codificarea perceptivă constă în a transforma sunetul digital, astfel încât un ascultător uman să nu poată face diferența între un sunet transformat și unul netransformat. La baza acestui tip de codificare stă *psihoacustica*.

Psihoacustica este știința care se ocupă de modul în care sunetele sunt percepute de către om.

Proprietatea de bază a *codificării perceptive* este constituită de faptul că unele sunete pot masca alte sunete, ast-

fel încât acestea din urmă nu se vor mai auzi atâta timp cât primele continuă să fie prezente. Pe baza acestei proprietăți, sunetele care nu vor putea fi auzite de ascultătorii umani vor fi eliminate.

În realitate, dacă generăm un sunet cu frecvența de 1 KHz la un volum de 60 dB și un alt sunet cu frecvența de 1.1 KHz și un volum de 40 dB, nu vom auzi cel de-al doilea sunet. Acest fenomen poartă numele de *mascarea frecvenței*. Sunetele care se aud poartă numele de *sunete mască*.

Un alt fenomen care are loc în realitate este acela că, dacă un sunet maschează un alt sunet, urechea umană este adaptată să audă numai sunetele mască. În cazul în care sunetele mască de la un moment dat dispar, celelalte sunete nu vor putea fi auzite decât după un inverval de timp, care reprezintă timpul în care urechea umană se reglează pentru a le putea percepe. Acest fenomen poartă numele de *mascare temporară*.

Prin urmare, o aplicație de compresie a sunetelor care se bazează pe *psihoacustică* va elimina pe baza unor relații sunetele care nu vor putea fi auzite de către om.

După eliminarea sunetelor care nu pot fi percepute asupra rezultatului se aplică unul dintre algoritmii de compresie fără pierdere de informație.

Se observă în acest caz că pierderea de informație este dată de eliminarea sunetelor care nu pot fi auzite de către om.

Raportul *calitate sunet/dimensiune fișier de ieșire* este mai mare dacă se aplică aceste reguli decât în cazul în care se încearcă compresia sunetelor fără a elimina sunetele care nu se aud.

Compresia imaginilor

Pentru a comprima imagini există mai multe metode. Unele aplicații comprimă imaginile fără a pierde informație.

De obicei, pentru a comprima o imagine, aceasta se împarte în mai multe imagini mai mici de formă pătrată sau dreptunghiulară.

În continuare vom prezenta modul de compresie a imaginilor folosit pentru a obține fișiere cu formatul *JPEG* (*Joint Photographic Expert Group*).

Primul pas în compresia imaginii constă în a descompune fiecare punct din imagine în componente *YIQ* (*Y* - luminanța; *I*, *Q* - crominanțe) sau *YUV* (*Y* - luminanța; *U*, *V* - crominanțe). Diferența între *YIQ* și *YUV* este dată de modul de calcul.

Componentele *Y*, *I*, *Q* se calculează pentru fiecare punct al imaginii astfel:

```
Y = 0,30·R + 0,59·G + 0,11·B
I = 0,60·R - 0,28·G - 0,32·B
Q = 0,21·R - 0,52·G + 0,31·B,
```

unde *R*, *G*, *B* sunt valorile pentru culorile roșu, verde și albastru.

Valorile componentelor *Y*, *I* și *Q* se păstrează în trei matrice distincte.

Al doilea pas constă în descompunerea fiecărei matrice în pătrate cu latura de 8 puncte.

Asupra fiecărui pătrat se aplică transformarea *DCT* bidimensională (*Discrete Cosine Transform - Transformarea Cosinus Discretă*).

După aplicarea transformării, pentru fiecare pătrat se obține o matrice *F* cu 8 linii și 8 coloane. Valorile din matricea *F* trebuie să fie numere întregi.

Valorile matricei *F* se cuantifică prin împărțirea întreagă la un număr întreg. Cuantificarea unei matrice *F* are ca scop reducerea numărului de biți necesari pentru a stoca informații despre imaginea care se comprimă.

Principala sursă a pierderii de informație în cazul formatului *JPEG* o constituie cuantificarea matricelor *F*.

Cuantificarea se poate face cu o constantă pentru toate valorile din matricea *F* (*cuantificare uniformă*). Constanta este diferită în cazul în care este vorba de o matrice *F* obținută prin aplicarea transformării *DCT* asupra unui pătrat obținut din matricea de luminanță și asupra unui pătrat obținut din una dintre matricele de crominanță. O altă modalitate de a cuantifica valorile matricei *F* este aceea de a utiliza două matrice de constante (*cuantificare neuniformă*): una pentru funcțiile *F* obținute din matricea de luminanță și alta pentru funcțiile *F* obținute din matricele de crominanță.

După cuantificarea fiecărei matrice *F* obținute prin împărțirea întreagă a fiecărui element din matricea *F* la elementul corespunzător din matricea constantă, rezultatul se depune într-un tablou unidimensional cu 64 de elemente în zig-zag, după cum indică figura alăturată.

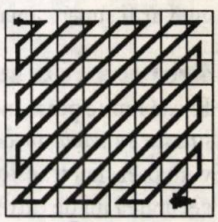


Figura 1

În continuare, tablourile obținute din matricea de luminanță se codifică folosind algoritmul *DPCM* (*Differential Pulse Code Modulation* - transformarea delta este un caz particular al *DPCM*), iar tablourile obținute din matricele de crominanță se codifică folosind algoritmul *RLE*.

Rezultatele obținute la pasul anterior se comprimă folosind algoritmul lui *Huffman* varianta statică sau semistatică.

În final, la ieșire se transmit constantele de cuantificare, codurile *Huffman* utilizate dacă acestea nu sunt standard (în cazul în care nu s-a folosit varianta statică cu coduri predefinite standard) și datele comprimate.

Bibliografie

1. **T. H. Cormen, C. E. Leisserson, R. R. Rivest,** *Introducere în algoritmi*, Computer Libris Agora, Cluj-Napoca, 2000
2. **A. S. Tanenbaum,** *Rețele de calculatoare*, ediția a patra, Byblos, București 2003
3. **J. A. Storer,** *An Introduction to Data Structures and Algorithms*, Springer, 2002
4. **http://www.datacompression.info/**
5. **http://www.researchindex.com/**

Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

TEOREME de programare

Clara Ionescu

Pe parcursul a opt luni (aproape a trecut un an școlar) v-am prezentat modele de proiectare și implementare a unor subalgoritmi care apar frecvent în rezolvarea problemelor. Aceste modele sugerează că programarea ar trebui să se desfășoare planificat, disciplinat, respectând anumite cerințe, pentru că altfel se diminuează beneficiul ei și vor fi înțelese numai de autorii lor.

Interclasarea a două șiruri ordonate

În numărul precedent am rezolvat problema reuniunii a două mulțimi, presupunând că mulțimile sunt implementate sub forma unor șiruri. Se poate observa ușor că, în cazul unor șiruri de date ordonate, această reuniune se va realiza altfel, mai ales dacă se cere ca și rezultatul să constituie un șir ordonat. De exemplu, fie problema:

P27. Se consideră mai multe tabele nominale ale elevilor din diferite școli, participanți la olimpiada județeană. Să se realizeze un singur tabel nominal în care să fie cuprinse numele tuturor elevilor din tabelele date!

În această problemă se dau mai multe șiruri ordonate și se cere unul singur, ordonat și acesta. Rezolvarea revine la crearea unui șir ordonat din *două* șiruri, apoi din șirul rezultat și următorul dat se va obține unul nou etc. În cele ce urmează vom prezenta un subalgoritm general care creează un șir ordonat din alte două șiruri ordonate. Deocamdată facem abstracție de enunțul problemei P27 în care, elementele șirurilor sunt cu siguranță distincte.

n, m: *întreg* {*numărul elementelor șirurilor date*}
X, Y: *tablou(1..n): tip element* { *elementele șirurilor date*}
buc: *întreg* { *numărul elementelor șirului nou*}
Z: *tablou(1..n): întreg* { *șirul nou obținut prin interclasare* }

În rezolvare vom avansa în paralel în cele două șiruri date. Primul element în șirul nou va fi fie `X[1]`, fie `Y[1]`. Pentru a ști care dintre cele două elemente va fi ales, trebuie să le comparăm. Dacă `X[1]` este mai mic decât `Y[1]`, atunci pe acesta îl "consumăm" din șirul *X* și pregătim indicele elementului următor din acesta. În caz alternativ, procedăm la fel cu indicele *j*, după ce așezăm elementul `Y[1]` în șirul *Z*. Dacă cele două elemente sunt egale, vom copia unul dintre aceștia în *Z*, și vom avansa în ambele șiruri. Vom efectua acești pași până când careva din șiruri se sfârșește, moment în care copiem elementele rămase din celălalt șir în șirul nou. Din moment ce nu putem ști care șir s-a "consumat" integral, vom scrie secvența respectivă pentru ambele șiruri.

```
subalgoritm Interclasare1(n,X,m,Y,bucZ,Z):
  buc ← 0
  i ← 1
  j ← 1
  cât timp (i <= n) și (j <= m) execută:
  buc ← buc + 1
    dacă X[i] < Y[j] atunci
      Z[buc] ← X[i]
      i ← i + 1
    altfel
      dacă X[i] = Y[j] atunci
        Z[buc] ← X[i]
        i ← i + 1
        j ← j + 1
      altfel
        Z[buc] ← y[j]
        j ← j + 1
      sfârșit dacă
    sfârșit dacă
  sfârșit cât timp
  cât timp i <= n execută:
  buc ← buc + 1
    Z[buc] ← X[i]
    i ← i + 1
  sfârșit cât timp
  cât timp j <= m execută:
  buc ← buc + 1
    Z[buc] ← Y[j]
    j ← j + 1
  sfârșit cât timp
sfârșit subalgoritm
```

Analizând acest subalgoritm, observăm că dacă am fi avut "norocul" ca `X[n]` să fie egal cu `Y[m]`, atunci ultimele două structuri **cât timp** nu se executau niciodată. Vom exploata această observație, așezând două elemente fictive (numite frecvent *santinele*) după ultimele elemente din cele două șiruri. Vom fi atenți să fie egale și să aibă valorile mai mari decât ultimul element (cel mai mare) din fiecare șir. În subalgoritmul `Interclasare2` vom nota această valoare cu `infinit`.

```
subalgoritm Interclasare2(n,X,m,Y,bucZ,Z):
  buc ← 0
  i ← 1
  j ← 1
  X[n+1] ← infinit
  Y[m+1] ← infinit
  cât timp (i < n+1) sau (j < m+1) execută:
  buc ← buc + 1
    dacă X[i] < Y[j] atunci
      Z[buc] ← X[i]
      i ← i + 1
    altfel
      dacă X[i] = Y[j] atunci
        Z[buc] ← X[i]
        i ← i + 1
        j ← j + 1
      altfel
        Z[buc] ← y[j]
        j ← j + 1
      sfârșit dacă
    sfârșit dacă
  sfârșit cât timp
sfârșit subalgoritm
```

Valoarea elementului fictiv (`infinit`) nu va face parte din șirul rezultat. Dacă am avea nevoie de acest element, pentru că șirul rezultat se va interclasa cu un altul, evident, putem să-l adăugăm.

Subalgoritmul `Interclasare2` este valabil în caz general. Să vedem cum s-ar aplica și ce îmbunătățiri s-ar putea realiza în scopul rezolvării unei probleme cum este P27, unde se știe că elementele sunt *distincte*. Evident, nu vom pune întrebarea privind posibila egalitate între `X[i]` și `Y[j]`. Dar mai știm ceva: șirul nou va conține fiecare element existent în șirul *X* și *Y*, deci se cunoaște numărul elementelor sale care este $n + m$, lucru care în caz general nu este cunoscut (se admit și elemente identice în cele două șiruri date). În acest caz, structura repetitivă utilizată va fi de tip **pentru**.

Rezultă că algoritmul de interclasare nu este "unul singur". În primul rând, dacă este posibil (și în majoritatea cazurilor este) vom adăuga elemente *santinele* la sfârșitul șirurilor care se vor interclasa, astfel eliberându-ne de grija elementelor rămase neprelucrate într-un șir, după ce celălalt s-a "epuizat". În al doilea rând, vom studia proprietatea elementelor din șirurile de interclasat: dacă ele sigur sunt distincte, vom putea implementa un algoritm care se prezintă sub următoarea formă:

```
subalgoritm Interclasare3(n,X,m,Y,bucZ,Z):
  i ← 1
  j ← 1
  X[n+1] ← infinit
  Y[m+1] ← infinit
  pentru buc = 1,n+m execută:
  dacă X[i] < Y[j] atunci
      Z[buc] ← X[i]
      i ← i + 1
    altfel
      Z[buc] ← y[j]
      j ← j + 1
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Concluzii

Ce s-ar mai putea spune? Este nevoie totuși de concluzii... În primul rând sperăm că se observă o tendință de reglementare a modului în care se vor implementa operații relativ simple care apar frecvent în algoritmi. Cu toții știm că este important să nu le realizăm neglijent. Aceste teoreme (*calcule pe baza elementelor unui șir* (1), *decizia* (2), *selecția* (3), *căutarea secvențială* (4), *numărarea* (5), *stabilirea maximului/minimului* (6), *selectarea* elementelor pe baza unei proprietăți (7), *partiționarea* (8), *reuniunea* (9), *intersecția* (10) și *interclasarea* (11)) conțin modele de implementare a unor astfel de operații "elementare".

Alte modele veți cunoaște în momentul studierii *Șabloanelor de programare* (*Design patterns*), atunci când veți decide să alegeți pentru viitor meseria de programator.

Rezultă că atunci când verificăm comportamentul și performanțele unui algoritm, nu va fi suficient să aplicăm o metodă de verificare de tipul "cutiei negre" sau "cutiei transparente". Astfel vom afla doar că algoritmul furnizează un rezultat "aparent" corect pentru date de intrare în cazul cărora am putut calcula și compara rezultatul (rezultatele) în minte. Dacă am obținut același rezultat în minte ca și cu ajutorul programului, ne putem bucura, dar nu putem fi siguri că algoritmul respectiv este cel mai bun (este optim) în ceea ce privește performanțele sale.

În momentul evaluării performanțelor trebuie să avem în vedere cele două resurse la care apelăm: *timpul* și *spațiul*. Vom evalua ordinul de mărime a numărului de operații care se vor efectua asupra datelor de prelucrat, apoi vom căuta să reducem acest număr. Vom încerca să renunțăm la structuri de date auxiliare, apelând la soluții care rezolvă problema "pe loc", adică folosind pentru rezultat structura de date construită pentru datele de intrare.

Bibliografie

1. Szalávi Péter, Zsakó László, *Programozási tételek*, Neumann János Számítógép - tudományi Társaság, 2001

GHIDUL TĂU ÎN LUMEA TEHNOLOGIILOR

Misiunea noastră este de a testa și evalua produse și servicii din domeniul tehnologiei informației și comunicațiilor și de a prezenta rezultatele cu onestitate și obiectivitate

AGORA MEDIA Str. C-tin Rădulescu Motru nr. 13, etaj 4, apt. 42, sect. 4 (CP 94/49), București
Tel:021-3309282, Fax:021-3309285

www.pcmagazine.ro

Agora MEDIA

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

FORTE COMPUTERS
Flexibility. Made in Romania

IBM®

sat
IT SOLUTIONS & SERVICES

hp®

net consulting

FUJITSU COMPUTERS SIEMENS

SPONSOR DE AUR

CCT
CABINETUL CĂLIN TATOMIR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

Octombrie 2003
Vol. 13/6

Lei 30.000

gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

Plus

soluții
ONI 2003-soluțiile problemelor propuse

probleme
Olimpiada Internațională de Informatică-2003

focus
Metode de abordare a problemelor NP

Premieră: Parteneriat strategic Microsoft - Guvernul României
PC MAGAZINE
ROMÂNIA
www.pcmagazine.ro
SOLUȚII
Discobolul
Primii pași în Linux
15 sfaturi pentru Excel
Cum să animezi o sigiă
TENDINȚE
Remodelarea PC-ului
Reportaj de la
Intel Developer Forum
Imagine
TESTE COMPARATIVE
Plăci video
PC LABS
GHIDUL TĂU ÎN LUMEA TEHNOLOGIILOR
Concurs de soluții!
Office
super premii
un PocketPC,
un SmartPhone,
sau un pachet
Office Professional 2003.
20
interfață grafică optimizată
conținut nou substanțial
încărcat 100%
Ghidul tău în lumea tehnologiei
o publicație Agora

Vol. 13, Nr. 6 /2003

gazeta de informatică

Σ sumar

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Director Bursele Agora:
Mihai Scorțaru
Redactor-șef:
Claudiu Soroiu
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Mihai Stroe

Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
400091, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/592.422, 0265/266.516
Fax: 0265/266.290
http://www.ginfo.ro
ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 0265/262.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
540014 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihai Bucuroiu
Cosmin Ursu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
Florin Oprescu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

probleme.. PAG. 5

Olimpiada INTERNAȚIONALĂ de informatică
~ IOI 2003 ~
~ Kenosha, S.U.A ~

Olimpiada BALCANICĂ de informatică
~ BOI 2003 ~
~ Iași, România ~

Olimpiada de informatică a EUROPEI CENTRALE
~ CEOI 2003 ~
~ Münster, Germania ~

Concursul INTERJUDEȚEAN de programare GRIGORE MOISIL
~ Satu Mare, martie 2003 ~

soluții.. PAG. 25

Olimpiada NAȚIONALĂ de INFORMATICĂ, ediția 2003
~ Soluțiile problemelor ~
~ propuse spre rezolvare ~
~ la clasele a IX-a și ~
~ a X-a ~

concurs.. PAG. 30

Regulamentul de participare la ediția 2003/2004 a concursului

Etapa finală a concursului internațional de programare "Bursele Agora"

focus.. PAG. 37

Metode de ABORDARE a problemelor NP
Mihai STROE

Drumuri MINIME când costurile sunt MICI
Mircea DIGULESCU
Andrei MATEI

babel.. PAG. 43

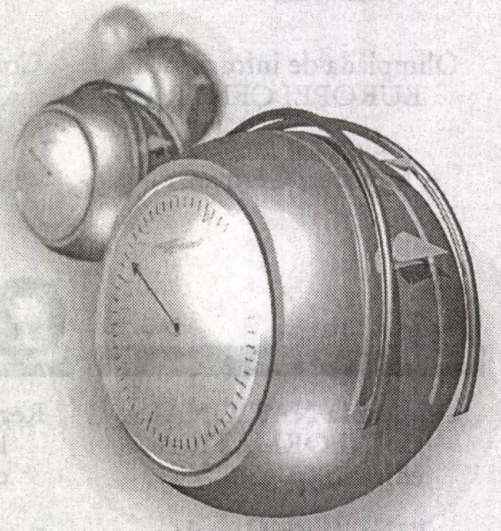
Conceptul de ALGORITM - abordare MODERNĂ
Marin VLADA
~ despre algoritmi ~
~ partea a treia ~

COMPUTER LIBRIS AGORA

DAN GRIGORAȘ

CALCULUL PARALEL

De la sisteme la programarea aplicațiilor



AGORA

Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA

COMPUTER LIBRIS AGORA

Informații la tel. 0264 - 592422

JOCUL și INFORMATICA

Pentru început țin să urez din partea redacției *GInfo* un sincer *Bun venit și succes în noul an!* tuturor elevilor și studenților care au intrat în noul an școlar.

Un nou an a început și în cadrul redacției noastre care acum se prezintă într-o configurație schimbată.

Nu puțini dintre elevii de la secțiile de informatică, pasionați de acest domeniu, au încercat să realizeze jocuri sau mai bine zis *joculețe*, la puțin timp după ce li s-au explicat principiile de bază ale algoritmicii și ale programării. Din nefericire, nu poți să-ți dai seama de rolul important pe care acest gen de activitate o are în dezvoltarea imaginației și structurarea gândirii, decât dacă ai reușit să finalizezi cel puțin un astfel de program.

Prin realizarea de jocuri, un elev din clasele mai mici are foarte multe de învățat și descoperit. Multe lucruri pe care un elev le descoperă singur i se par uluitoare, deși, pentru cei mai mari și interesați de informatică, lucrurile stau cu totul altfel.

Unora dintre acești elevi le-au fost puse *bețe-n roate* de diferite persoane. De exemplu, elevul, încântat, le povestește părinților ce a realizat, iar o parte dintre ei au replici de genul "*Nu mai face prostii, fă și tu ceva util!*".

Dacă vi se pare că nu aș avea dreptate în ceea ce privește faptul că realizarea unui joc este *utilă* pentru dezvoltarea modului de gândire, vă invit să citiți mai departe.

O categorie însemnată de persoane care consideră că elevul care implementează jocuri pe calculator ar avea și altceva mai bun de făcut, sunt chiar profesorii de informatică. Unora le voi da dreptate în privința unui singur fapt, și anume, acela că elevul trebuie să-și pregătească mai întâi temele pentru orele de la școală și abia după aceea să-și continue activitatea creativă (sau invers, ordinea nu contează ci numai prioritatea).

În momentul în care un profesor de informatică observă că un anumit elev este pasionat de informatică încearcă să-l pregătească pentru concursurile de informatică. După cum știți, o foarte mică parte a elevilor pasionați de informatică sunt interesați de concursuri. Unii dintre ei își cunosc limitele, iar alții își subestimează forțele proprii.

În cazul elevilor care nu au devenit interesați de concursurile de informatică, încercarea de a-i face să învețe algoritmi noi nu duce decât la încetinirea procesului de învățare pentru aceștia. Desigur este important și modul cum se încearcă apropierea elevilor de concursurile de informatică.

Mediul social în care trăiește un elev este, de asemenea, unul dintre factorii care duc la încetarea procesului de învățare prin realizarea de jocuri. Ce părere aveți de următorul dialog, purtat de doi colegi de clasă:

- *Aseară am făcut un joc de șah.*
- *Și poți să joci cu programul tău?*
- *Da, dar îl bat cam de fiecare dată.*
- *Ce aiurea, cu programul făcut de tine nu poți învăța șah, și majoritatea programelor de acum joacă foarte bine. La ce folos încă unul?... Eu aseară am jucat jocul XYZ și am avut cele mai multe puncte dintre toți.*

În urma unui astfel de dialog, realizatorul jocului de "*șah*" din păcate, își va pierde interesul pentru informatică, domeniu vast, plin de surprize și care are nevoie din ce în ce mai mare de mai mulți programatori buni.

Vă mărturisesc că eu am realizat în trimestrul al doilea din clasa a *IX*-a un joc de *tetris* în *Pascal*. Am ținut cu tot dinadinsul să-l fac în mod text, pentru că mi-a plăcut mie un joc asemănător pe care l-am văzut la un prieten și piesele erau formate din jumătăți de pătrățele (nu din caracterul ■, ci din caracterele ▬ și ▀).

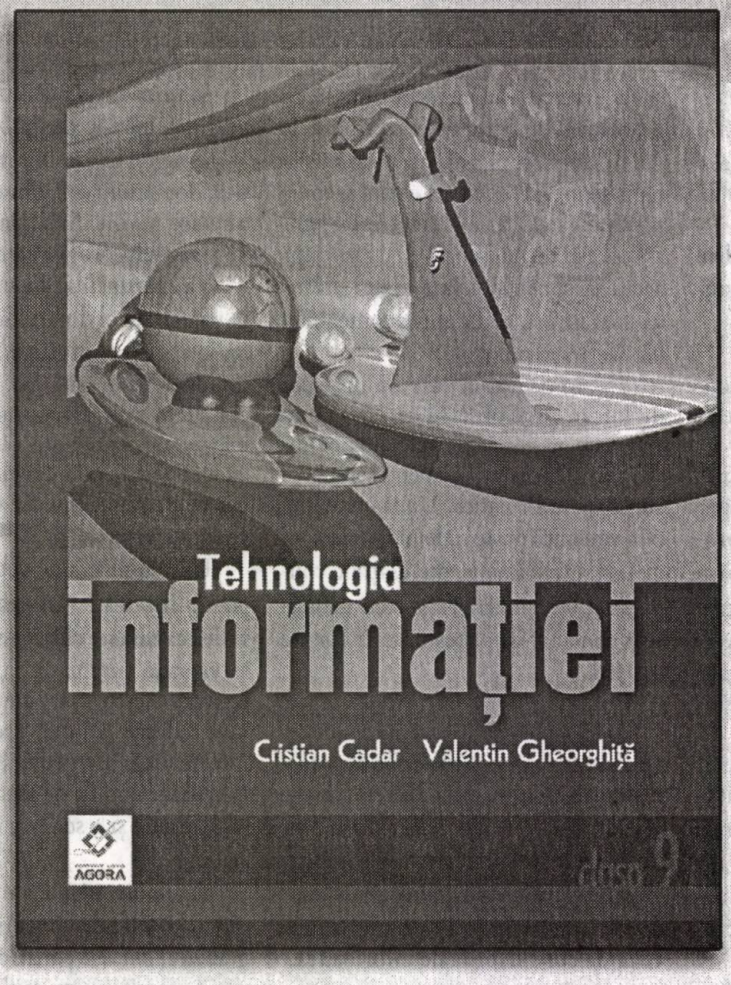
Unii dintre voi veți zice "*Simplu!*". Încercați să-l realizați și veți vedea că nu este chiar așa *simplu*. Poate va suna surprinzător, dar din realizarea acestui joc am învățat mai multe decât la orele propriu-zise de informatică: elemente de grafică pe calculator, limbaj de asamblare, structuri de date și o serie de detalii interesante.

Recunosc că nu am fost atras în mod deosebit de concursurile de programare, dar am participat la ele și cu puțin noroc și cu multe cunoștințe acumulate din "jocuri" am reușit să rezolv și probleme pe care nu le-am mai întâlnit niciodată. Dar, cum v-am mai spus: lumea nu se învârtește în jurul concursurilor...

Am convingerea că prin realizarea de "*jocuri*" un elev învață să gândească *algoritmic* (structurat și logic) și mai târziu, îi va fi ușor să atace cu curaj și să rezolve elegant probleme reale, pe scurt, să devină un programator apreciat.

Claudiu Soroiu

COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel.0264 - 592422

COMPUTER LIBRIS AGORA

Kenosha, Statele Unite ale Americii

IOI 2003

În perioada 16-23 august 2003 a avut loc cea de-a XV-a ediție a concursului internațional de programare IOI. Acesta a fost găzduit de Universitatea Wisconsin Parkside, din Statele Unite ale Americii.

Ediția de anul acesta a *Olimpiadei Internaționale de Informatică* a fost găzduită de *Universitatea Wisconsin Parkside*, situată în orașul american *Kenosha*.

Sponsorul oficial al acestei competiții a fost compania *Microsoft*.

În acest an echipa României a fost reprezentată de patru concurenți. Aceștia sunt:

Victor-Marius Costan, București
Radu Berinde, București
Dan Ghinea, București
Ștefan Ciobâcă, Suceava

Victor-Marius Costan și *Radu Berinde*, obținând 424 de puncte, respectiv 353,9 puncte, au câștigat medalii de aur. Ceilalți doi concurenți români, *Ștefan Ciobâcă* și *Dan Ghinea*, obținând 289,9 puncte, respectiv 366,4 puncte, au câștigat medalii de argint.

Cu aceste rezultate obținute de echipa țării noastre, România a ocupat locul I în clasamentul pe echipe, alături de *Statele Unite ale Americii*, *Coreea* și *Suedia*.

Nici unul dintre participanții la ediția din acest an a *Olimpiadei Internaționale de Informatică* nu a reușit să obțină punctajul maxim, și anume 600.

Cel mai mare număr de puncte a fost obținut de către concurentul coreean *Hwan-Seung Yeo* care a acumulat cu rezolvările sale 455,4 puncte, aflându-se la o diferență de aproximativ 20 de puncte de următorul.

Concurenților li s-au pus la dispoziție sisteme dotate cu procesoare *Intel Pentium* 4 cu o frecvență de 2,2 *GHZ*, 256 MB memorie RAM și unități *CD-RW* pe care au fost instalate sistemele de operare *Microsoft Windows XP Professional* și *Linux RedHat* 9. În timpul concursului, participanților li s-a permis să folosească propriile tastaturi, dispozitive de indicare (de exemplu, alt *mouse*) și propriile *CD*-uri pentru a face copii de siguranță.

În timpul evaluării programele concurenților au avut la dipoziție 64MB de memorie *RAM*.

În continuare vă prezentăm enunțurile celor șase probleme propuse spre rezolvare.

P060301: Comparări de cod

Compania *Racine Business Networks* (*RBN*) a dat în judecată compania *Heuristic Algorithm Languages* (*HAL*), invocând faptul că *HAL* a preluat o parte din codul sursă *RBN UNIX™* și l-a inclus în sistemul de operare cu fișierele sursă deschise *HALnix*.

Ambele companii *RBN* și *HAL* utilizează un limbaj de programare cu câte o instrucțiune pe linie de forma: `STOREA = STOREB + STOREC`, unde `STOREA`, `STOREB` și `STOREC` sunt nume de variabile.

Primul nume de variabilă începe din prima coloană, după el urmează un spațiu, semnul egal, un alt spațiu, al doilea nume de variabilă, iarăși un spațiu, simbolul plus, din nou un spațiu și al treilea nume de variabilă. Același nume de variabilă poate apărea pe o singură linie mai mult decât o singură dată. Numele de variabile sunt formate din cel puțin una și cel mult 8 litere mari ale alfabetului englez.

Compania *RBN* pretinde că *HAL* a copiat direct din codul sursă elaborat de *RBN* o anumită secvență de linii consecutive, făcând numai modificări minore:

- Compania *RBN* pretinde că în scopul tăinuirii infracțiunilor sale, compania *HAL* a schimbat unele nume de variabile. Mai exact, compania *HAL* a preluat o secvență de linii din programul elaborat de compania *RBN* și pentru fiecare variabilă din această secvență a schimbat toate aparițiile numelui respectiv cu un nume nou, uneori numele nou putând să coincidă cu cel inițial. Desigur, variabilele distincte din programul inițial au denumiri distincte și în programul final.
- De asemenea, compania *RBN* pretinde că *HAL* e posibil să fi modificat ordinea de apariție a variabilelor din partea dreaptă a semnului egal din unele linii, de exemplu, linia `STOREA = STOREB + STOREC` posibil să fi fost modificată în `STOREA = STOREC + STOREB`.
- Compania *RBN* afirmă că *HAL* nu a modificat ordinea liniilor din codul sursă propus de *RBN*.

Elaborați un program, care, având ca date inițiale codurile sursă ale ambelor companii, găsește lungimea celei mai lungi secvențe de linii consecutive din programul elaborat de compania *HAL*, care poate fi obținută dintr-o secvență de linii consecutive din programul *RBN*, utilizând transformările descrise mai sus. Menționăm că secvențele de linii consecutive din cele două programe nu trebuie să înceapă în aceeași poziție în cadrul programelor.

Date de intrare

Fișierul de intrare **code.in** conține pe prima linie două numere întregi pozitive R și H, separate prin spațiu, care reprezintă numărul de linii de cod ale programului sursă *RBN*, respectiv numărul de linii de cod ale programului *HAL*. Pe următoarele R linii se află programul *RBN*, iar în continuare, pe următoarele H linii se află programul *HAL*.

Date de ieșire

Fișierul de ieșire **code.out** va conține un singur număr întreg pozitiv care reprezintă lungimea celei mai lungi secvențe de linii consecutive din programul *RBN* pe care compania *HAL* le-ar fi copiat și transformat.

Restricții

- $1 \le R \le 1000$;
- $1 \le H \le 1000$.

Exemplu

code.in

```
4 3
RA = RB + RC
RC = D + RE
RF = RF + RJ
RE = RF + RF
HD = HE + HF
HM = HN + D
HN = HA + HB
```

code.out

```
2
```

Explicație

Liniile 1 și 2 ale programului *RBN* conicid cu liniile 2 și 3 ale programului *HAL*, dacă numele de variabile sunt înlocuite după cum urmează: RA → HM, RB → D, RC → HN, D → HA, RE → HB. Nu există secvențe de trei sau mai multe linii care ar fi compatibile.

Timp de execuție: 2 secunde/test

P060302: Întreținerea potecilor

Vacile fermierului *John* doresc să umble libere pe cele N pășuni din fermă, numerotate de la 1 la N, chiar dacă pășunile sunt separate prin păduri. Vacile doresc să întrețină potecile existente între perechi de pășuni astfel încât să poată parcurge drumul care leagă oricare pășune de oricare alta, utilizând potecile întreținute. Vacile pot umbla pe o potecă întreținută în ambele sensuri.

Vacile nu construiesc poteci. În schimb, ele întrețin potecile animalelor sălbatice pe care le-au descoperit. În orice săptămână ele pot să aleagă să întrețină unele sau toate potecile pe care le cunosc.

Curios, la începutul fiecărei săptămâni, vacile descoperă o nouă potecă de animale sălbatice. După aceea trebuie să decidă mulțimea de poteci pe care le vor întreține în săptămâna respectivă astfel încât să poată parcurge drumul de la orice pășune la orice alta. La un moment dat, vacile nu pot folosi decât potecile pe care le întrețin în momentul respectiv.

Vacile doresc întotdeauna să minimizeze lungimea totală a potecilor pe care trebuie să le întrețină. Vacile pot să aleagă pentru întreținere orice submulțime de poteci de animale sălbatice pe care le cunosc, indiferent de potecile pe care le-au menținut în săptămâna precedentă.

Potecile de animale sălbatice (chiar și atunci când sunt întreținute) nu sunt niciodată drepte.

Două poteci care unesc aceleași două pășuni pot avea lungimi diferite. Chiar dacă două poteci se intersectează, vacile sunt atât de concentrate încât refuză să schimbe poteca, exceptând cazul în care sunt pe o pășune.

La începutul fiecărei săptămâni, vacile vă vor descrie poteca de animale sălbatice pe care au descoperit-o. Programul vostru trebuie să afișeze lungimea totală minimă a potecilor pe care vacile trebuie să le întrețină în săptămâna respectivă, astfel încât să poată parcurge drumurile care între oricare două pășuni, dacă există o astfel de mulțime de poteci.

Date de intrare

Datele se citesc de la intrarea standard. Prima linie va conține două numere întregi N și W, separate printr-un singur spațiu, care reprezintă numărul de pășuni din fermă, respectiv numărul de săptămâni pe care programul le va rezolva.

Pentru fiecare săptămână, datele se dau pe o singură linie care conține informații despre poteca de animale sălbatice descoperită. Această linie conține trei numere întregi separate prin spații: numerele de ordine ale celor două pășuni care reprezintă extremitățile potecii) și lungimea potecii.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. În momentul în care programul vostru află date despre noua potecă de animale sălbatice descoperită trebuie să afișeze o singură linie care conține lungimea totală minimă a potecilor pe care vacile trebuie să le întrețină astfel încât să poată parcurge drumurile care leagă oricare pășune la oricare alta.

Dacă nici o mulțime de poteci nu permite vacilor să traverseze drumurile între oricare două pășuni, programul va afișa "-1".

Programul vostru trebuie să se termine după ce afișează răspunsul pentru ultima săptămână.

Restricții și precizări

- $1 \le N \le 200$;
- $1 \le W \le 6000$;
- lungimile potecilor sunt numere întregi cuprinse între 1 și 10000;
- nici o potecă de animale sălbatice nu are aceeași pășune în ambele extremități.

Exemplu

Intrare	*Ieșire*
4 6	
1 2 10	-1
1 3 8	-1
3 2 3	-1
1 4 3	14
1 3 6	12
2 1 2	8

Explicație

După descoperirea primei poteci, mulțimea de poteci va fi formată dintr-un singur element, deci nici o potecă din mulțime nu leagă pășunea 4 de restul pășunilor.

Nici după descoperirea celei de-a doua și a treia poteci, în mulțime nu va exista o potecă ce leagă pășunea 4 de restul pășunilor.

După descoperirea celei de-a patra poteci, vor trebui întreținute potecile: 1 4 3, 1 3 8 și 3 2 3 a căror lungime totală este 14.

După descoperirea celei de-a cincia poteci, vor trebui întreținute potecile: 1 4 3, 1 3 6 și 3 2 3 a căror lungime totală este 12.

După descoperirea ultimei poteci, vor trebui întreținute potecile: 1 4 3, 2 1 2 și 3 2 3 a căror lungime totală este 8.

Timp de execuție: 1 secundă/test

P060303: Inversare

Să considerăm o mașină care execută două operații (pe scurt denumită *TOM*) și care are nouă regiștri, numerotați de la 1 la 9. Fiecare registru poate memora un număr natural din intervalul [0, 10000]. Mașina poate executa două operații:

- `S i j` - memorează în registrul `j` valoarea din registrul `i` la care se adaugă `1` și valoarea lui `i` poate fie egală cu valoarea lui `j`;
- `P i` - afișează valoarea memorată în registrul `i`.

Un program *TOM* include un set de valori inițiale ale regiștrilor și o secvență de operații. Fiind dat un număr natural *N*, generați un program *TOM* care să afișeze secvența descrescătoare de numere întregi *N*, *N* - 1, *N* - 2, ..., 0. Numărul maxim de operații `S` consecutive trebuie să fie cât mai mic posibil.

În continuare aveți un exemplu de program *TOM* și execuția lui pentru *N* = 2.

Operație	Valori regiștrilor	Valoarea afișată
Valori inițiale	0 2 0 0 0 0 0 0 0	
P 2	0 2 0 0 0 0 0 0 0	2
S 1 3	0 2 1 0 0 0 0 0 0	
P 3	0 2 1 0 0 0 0 0 0	1
P 1	0 2 1 0 0 0 0 0 0	0

Date de intrare

Aceasta este o problemă cu fișiere deschise. Fișierele de intrare sunt formate din două linii.

Prima linie a unui fișier de intrare conține un singur număr întreg *K*, reprezintă numărul testului, iar a doua linie conține numărul natural *N*.

La acest concurs au fost puse prin intermediul *server*-ului 16 fișiere de intrare.

Date de ieșire

Nu sunt restricții în ceea ce privește numele fișierelor de ieșire, dar acestea trebuie să respecte formatul descris în continuare.

Prima linie a unui fișier de ieșire trebuie să conțină șirul "`FILE reverse K`", unde `K` reprezintă numărul testului și este cuprins între 1 și 16.

A doua linie trebuie să conțină 9 numere separate între ele prin spațiu care reprezintă valorile inițiale ale regiștrilor, pe care le doriți, în ordine, adică primul număr reprezintă valoarea inițială a primului registru, cel de-al doilea număr reprezintă valoarea inițială a celui de-a doilea registru etc.

Restul fișierului de ieșire trebuie să conțină o listă ordonată de operații care trebuie executate, câte o operație pe linie.

Astfel, a treia linie conține prima operație care trebuie executată, și așa mai departe. Ultima linie a fișierului trebuie să fie cea care afișează valoarea 0.

Fiecare linie trebuie să conțină o operație validă. Operațiile trebuie să fie formatate ca în exemplu.

Restricții

- $0 \le N \le 255$;
- numărul de operații `S` consecutive nu trebuie să fie mai mare decât 131;
- o operație `S` nu trebuie să ducă la depășirea valorii maxime admise pentru un registru.

Exemplu

Conținut fișier de intrare

```
1
2
```

Conținut fișier de ieșire 1

```
FILE reverse 1
0 2 1 0 0 0 0 0 0
P 2
P 3
P 1
```

Conținut fișier de ieșire 2

```
FILE reverse 1
0 2 0 0 0 0 0 0 0
P 2
S 1 3
P 3
P 1
```

Modalitate de acordare a punctajului

Din punctajul maxim care putea fi obținut pentru un test s-a acordat 20% pentru corectitudinea fișierului de ieșire.

Un fișier de ieșire este corect dacă respectă formatul descris și, de asemenea, respectă restricțiile impuse.

Restul de 80% a fost acordat pentru optimalitatea programului *TOM* trimis.

Optimalitatea unui program *TOM* corect este măsurată de numărul maxim de operații `S` consecutive din program, care trebuie să fie cât mai mic posibil.

P060304: Frontiera

Fermierul *Don* se uită la gardul care înconjoară terenul său plat de formă pătrată de N metri pe N metri. Un colț al gardului este în origine (0, 0), iar colțul opus este în punctul de coordonate (N, N); laturile gardului fermierului *Don* sunt paralele cu axele X și Y.

Stâlpii gardului nu apar numai în cele patru colțuri, ci și la fiecare metru de-a lungul gardului, deci în total $4 \cdot N$ stâlpi. Stâlpii sunt verticali și considerați de rază neglijabilă. Fermierul *Don* vrea să determine câți stâlpi din gardul său poate vedea atunci când stă într-o poziție dată.

Terenul fermierului *Don* conține stânci uriașe care obturează vizibilitatea către anumiți stâlpi, deoarece el nu este suficient de înalt pentru a se uita peste aceste stânci.

Baza fiecărei stânci este un poligon convex cu arie nenulă ale cărui vârfuri sunt în puncte de coordonate întregi. Stâncile stau complet verticale. Stâncile nu se suprapun, nu ating alte stânci, nu îl ating pe fermierul *Don* și nici gardul.

Fermierul *Don* nu atinge gardul, nu stă într-o stâncă sau pe o stâncă.

Fiind dată dimensiunea gardului, poziția și forma stâncilor și poziția în care se află fermierul *Don*, determinați numărul stâncilor pe care fermierul *Don* le poate vedea.

Date de intrare

Prima linie a fișierului de intrare **`boundary.in`** conține două valori întregi N și R, separate printr-un singur spațiu, reprezentând lungimea laturii terenului, respectiv numărul de stânci care obturează vizibilitatea către anumiți stâlpi.

Cea de-a doua linie conține două numere întregi separate printr-un singur spațiu care reprezintă coordonatele X și Y ale fermierului *Don* din interiorul gardului.

Restul fișierului de intrare descrie cele R stânci, astfel:

- descrierea stâncii i începe cu o linie conținând un singur număr întreg p_i, reprezentând numărul de vârfuri ale bazei stâncii.
- fiecare dintre următoarele p_i linii conține o pereche de numere întregi, separate prin spațiu, reprezentând coordonatele X și Y ale unui vârf. Vârfurile bazei unei stânci sunt distincte și sunt date în sensul invers al acelor de ceasornic.

Date de ieșire

Fișierul de ieșire **`boundary.out`** conține o singură linie pe care se va afla un singur număr întreg care reprezintă numărul de stâlpi pe care îi poate vedea fermierul *Don* din locul în care se află.

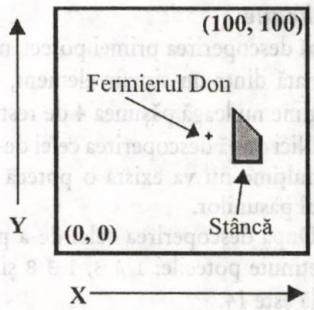
Restricții

- $2 \le N \le 500.000$;
- $1 \le R \le 30.000$;
- $3 \le p_i \le 20$, $1 \le i \le R$.

Exemplu

`boundary.in`

```
100 1
60 50
5
70 40
75 40
80 40
80 50
70 60
```



`boundary.out`

```
319
```

Timp de execuție: 1 secundă/test

P060305: Ghicește vaca

Cele N vaci din cireada fermierului *John* sunt foarte asemănătoare și sunt numerotate de la 1 la N. Când fermierul *John* duce o vacă la culcare în staulul ei, trebuie să determine despre care vacă este vorba ca să o pună în staulul corect.

Vacile sunt diferențiate utilizând P proprietăți, numerotate de la 1 la P, fiecare dintre ele având trei valori posibile. De exemplu, culoarea etichetei de pe urechea vacii poate fi *galben*, *verde* sau *roșu*. Pentru simplitate, valorile fiecărei proprietăți sunt desemnate prin literele `'X'`, `'Y'` și `'Z'`.

Oricare două vaci ale fermierului *John* diferă prin cel puțin o proprietate.

Scrieți un program care, plecând de la proprietățile vacilor din cireada fermierului *John*, determină care vacă trebuie dusă la culcare.

Programul poate pune fermierului *John* cel mult 100 de întrebări de forma:

"*Valoarea proprietății T a vacii aparține mulțimii S?*"

Încercați să puneți cât mai puține întrebări pentru a ghici care vacă va merge la culcare.

Date de intrare

Prima linie a fișierului de intrare **guess.in** conține două valori întregi N și P, separate printr-un singur spațiu, care reprezintă numărul de vaci, respectiv numărul de proprietăți.

Aceasta este o problemă interactivă.

Întrebările sunt puse prin intermediul ieșirii standard.

După punerea unei întrebări, programul trebuie să citească o singură linie care conține un singur număr întreg. Dacă numărul citit este egal cu 1 înseamnă că proprietatea specificată pentru vaca trimisă la culcare se află în mulțimea de valori dată, în caz contrar, dacă numărul citit este egal cu 0 înseamnă că proprietatea specificată nu se află în mulțimea de valori dată.

Date de ieșire

Programul trebuie să pună întrebări despre vaca trimisă la culcare scriind la ieșirea standard o linie care începe cu caracterul `'Q'` urmat de un spațiu, numărul proprietății, un spațiu și o mulțime de una sau mai multe valori separate prin spații.

De exemplu, `Q 1 Z Y` semnifică întrebarea "*Proprietatea 1 a vacii trimise la culcare este* `'Z'` *sau* `'Y'`*?*". Proprietatea trebuie să fie un număr întreg din intervalul [1, P].

Valorile trebuie să fie `'X'`, `'Y'` sau `'Z'` și nici o valoare nu trebuie să apară într-o întrebare de mai multe ori.

Ultima linie pe care o afișează programul trebuie să înceapă cu caracterul `'C'` urmat de un spațiu și apoi un singur număr întreg care specifică vaca pe care programul a determinat-o ca fiind cea pe care fermierul *John* o duce la culcare.

Restricții și precizări

- $1 \le N \le 50$;
- $1 \le P \le 8$;
- programul vostru nu trebuie să pună mai mult de 100 de întrebări.

Exemplu

guess.in

```
4 2
X Z
X Y
Y X
Y Y
```

Interactivitate

Întrebare: `Q 1 X Z`
Răspuns: `0`
Întrebare: `Q 2 Y`
Răspuns: `1`
Rezultat: `C 4`

Explicație

Răspunsul la întrebarea "*Proprietatea 1 a vacii care este trimisă la culcare este* `'X'` *sau* `'Z'`*?*" este *nu* (0), aceasta înseamnă că vaca nu poate avea numerele de ordine 1 și 2, deci trebuie să fie 3 sau 4.

Răspunsul la întrebarea "*Proprietatea 2 a vacii care este dusă la culcare este* `'Y'`*?*" este *da* (1), aceasta înseamnă că vaca poate avea numai numărul de ordine 4, deoarece proprietate 2 a vacii cu numărul de ordine 3 nu este `'Y'`.

După ce este dat răspunsul la ultima întrebare, se poate stabili că rezultatul corect este 4 și programul se oprește după scrierea acestuia la ieșirea standard.

Modalitate de acordare a punctajului

Din punctajul maxim care putea fi obținut pentru un test s-a acordat 30% pentru corectitudinea fișierului de ieșire. Programul a primit tot punctajul pentru corectitudine numai dacă vaca afișată a fost singura vacă ce corespundea răspunsurilor primite. Un program care punea mai mult de 100 de întrebări într-un test nu a primit puncte pe acel test.

Restul de puncte au fost determinate de numărul de întrebări necesare pentru determinarea corectă a vacii. Modul de acordare a punctelor i-a recompensat pe cei care au minimizat numărul de întrebări în cazul cel mai defavorabil.

Timp de execuție: 1 secundă/test

P060306: Roboții uimitori

Sunteți posesorul a doi roboți care se află în labirinturi dreptunghiulare separate. Coordonatele (1, 1) indică pătratul din colțul stânga-sus sau, cu alte cuvinte, din colțul nord-vest al labirintului. În labirintul i ($i \in \{1, 2\}$) există G_i gardieni, care se străduie să prindă robotul respectiv. Fiecare gardian se deplasează continuu mai întâi într-o direcție, iar apoi în cea opusă, de-a lungul unui segment de dreaptă, care reprezintă un traseu prestabilit de patrulare. Scopul vostru este de a determina o secvență de comenzi după execuția cărora roboții vor fi scoși din labirinturile respective fără ca ei să fie capturați de gardieni.

În fiecare minut voi transmiteți ambilor roboți aceeași comandă. Fiecare comandă reprezintă o direcție de deplasare (*Nord*, *Sud*, *Est*, *Vest*). Execuția unei comenzi presupune deplasarea robotului cu un pătrățel în direcția indicată. Dacă în direcția indicată se află un perete, în minutul respectiv robotul rămâne nemișcat. Se consideră că robotul a ieșit din labirint atunci când el se află în exteriorul lui. După ieșirea din labirint, robotul ignoră toate comenzile ulterioare.

În fiecare minut, concomitent cu roboții, gardienii se mută cu un pătrat. Fiecare gardian pornește din pătratul inițial, se deplasează continuu în direcția indicată cu viteza de un pătrat pe minut până când lungimea drumului parcurs este egală cu cea a traseului de patrulare minus unu. În continuare, gardianul se întoarce instantaneu și se deplasează în direcția opusă, către locația de start, unde se întoarce din nou și iarăși repetă traseul de patrulare ș.a.m.d. până când fiecare robot va fi scos din labirintul său. Patrularea nu presupune trecerea gărzii prin pereți sau ieșirea ei din labirint. Deși traseele de patrulare se pot suprapune,

gardienii nu se ciocnesc niciodată, adică, în orice minut, nici o pereche de gardieni nu se va afla în unul și același pătrat sau nu vor face un schimb de pătrate adiacente. Inițial, orice gardian din fiecare labirint se află într-un pătrat care nu coincide cu cel în care se află robotul.

Gardianul capturează un robot atunci când, în minutul curent, ambii se află în același pătrat sau când ei schimbă pătratele adiacente.

Cunoscând cele două labirinturi (fiecare din ele nu mai mare de 20×20), pătratele inițiale ale fiecărui robot și traseele de patrulare ale gardienilor, calculați o secvență de comenzi după execuția căreia roboții vor fi scoși din labirinturi fără ca ei să fie capturați de gardieni.

Minimizați timpul necesar pentru ca ultimul dintre roboți să fie scos din labirint. Dacă roboții ies din labirinturile respective în momente de timp diferite, momentul ieșirii primului dintre ei nu contează.

Date de intrare

Numele fișierului de intrare este **robots.in**. Primul set de linii descrie primul labirint, robotul și gărzile din labirint. Într-un mod similar, setul al doilea de linii descrie labirintul al doilea, robotul și gărzile din el.

Prima linie din fișierul de intrare conține două numere întregi R_1 și C_1, separate prin spațiu, care reprezintă numărul de rânduri și numărul de coloane ale primului labirint.

Fiecare dintre următoarele R_1 linii conține câte C_1 caractere care descriu configurația labirintului. Poziția inițială a robotului este redată prin simbolul `'X'`. Simbolul `'.'` reprezintă un pătrat liber, iar simbolul `'#'` reprezintă un perete. În fiecare labirint se află câte un singur robot.

După descrierea labirintului urmează o linie care conține numărul întreg G_1, care indică numărul de gărzi din primul labirint.

Fiecare dintre următoarele G_1 linii descrie un traseu de patrulare, folosind în acest scop trei numere întregi și un caracter, separate printr-un singur spațiu. Primele două numere întregi indică rândul și coloana pătratului de start al gărzii. Numărul al treilea indică numărul de pătrățele (2..4) din traseul de patrulare. Caracterul indică direcția inițială în care se va mișca gardianul: `'N'`, `'S'`, `'E'`, `'W'` (*Nord*, *Sud*, *Est*, *Vest*).

Date de ieșire

Prima linie din fișierul de ieșire **robots.out** va conține un singur număr întreg K, reprezentând numărul de comenzi după execuția cărora roboții vor fi scoși din labirint fără ca ei să fie capturați.

Dacă o astfel de secvență există, cea mai scurtă va avea nu mai mult de 10000 de comenzi.

Următoarele K linii vor descrie secvența de comenzi, fiecare linie conținând câte un caracter din mulțimea {`'N'`, `'S'`, `'E'`, `'W'`}.

Dacă o astfel de secvență de comenzi nu există, scrieți la ieșire o singură linie care conține `"-1"`.

Ambii roboți ar trebui să iasă din labirinturi după execuția ultimei comenzi. Ultima comandă ar trebui să scoată din labirint cel puțin unul dintre roboți. Dacă există mai multe secvențe de comenzi care scot roboții din labirinturi într-un timp minim, va fi acceptată oricare din ele.

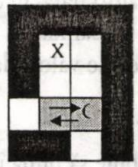
Restricții și precizări

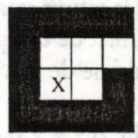
- $1 \le G_1, G_2 \le 10$;
- $1 \le K \le 10.000$.

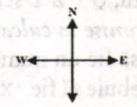
Exemplu

robots.in

```
5 4
####
#X.#
#..#
...#
##.#
1
4 3 2 W
####
#...
#X.#
####
0
```







robots.out

```
8
E
N
E
S
S
S
E
S
```

Modalitate de acordare a punctajului

Pentru testele în care secvența de comenzi nu exista, nu s-a acordat punctaj parțial. În celelalte cazuri, punctajul parțial s-a acordat după cum urmează.

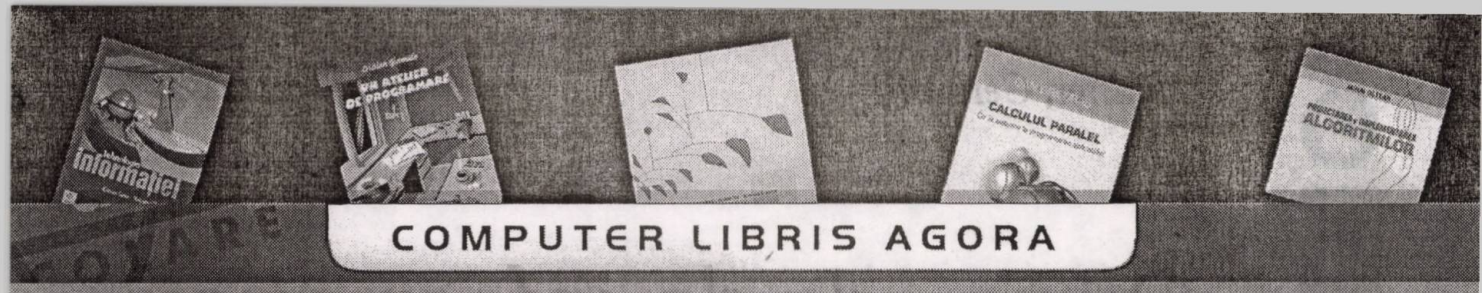
Din punctajul maxim care putea fi obținut pentru un test s-a acordat 20% pentru corectitudinea fișierului de ieșire.

Fișierul de ieșire pentru un test s-a considerat corect, dacă acesta a fost formatat corect, conținea cel mult 10000 de comenzi, secvența respectivă de comenzi scotea roboții din labirinturi, iar ultima comandă din secvență scotea cel puțin un robot.

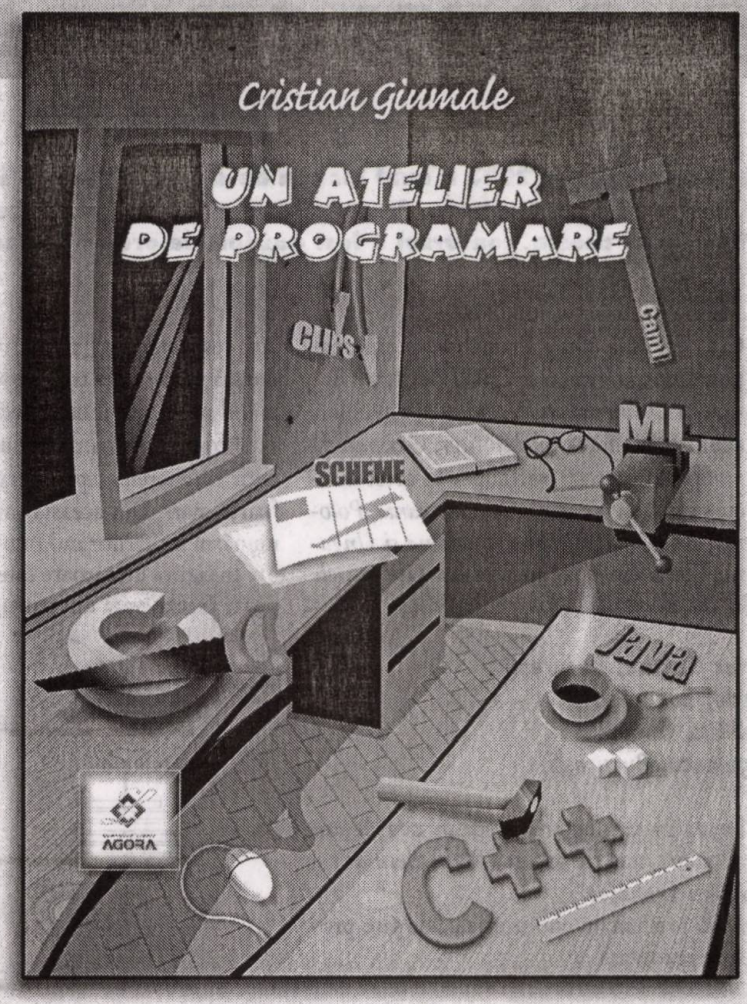
Restul de puncte s-au acordat pentru minimalitatea fișierului de ieșire. Fișierul de ieșire pentru un test s-a considerat a fi minimal, dacă acesta a fost corect și nu exista o altă secvență corectă de comenzi care să fi fost mai scurtă.

Nu s-au acordat puncte pentru minimalitate în cazul secvențelor care nu erau cele mai scurte.

Timp de execuție: 2 secunde/test



COMPUTER LIBRIS AGORA



Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS AGORA

Münster, Germania

Olimpiada de informatică a EUROPEI CENTRALE

În perioada 5-12 iulie 2003 a avut loc cea de-a X-a ediție a Olimpiadei de Informatică a Europei Centrale. Acest concurs a fost inițiat de țara noastră în anul 1994. Anul acesta, concursul a fost găzduit de orașul german Münster.

Cea de-a X-a ediție a concursului internațional de programare *CEOI* (*Central European Olympiad in Informatics*) a avut loc, anul acesta, în orașul *Münster, Germania.*

Țările care au participat la acest concurs sunt: *Cehia, Croația, Germania, Polonia, România, Slovacia, Slovenia* și *Ungaria.* De asemenea, au fost invitate *Statele Unite* și *Iran.* Echipa *României* a fost reprezentată de:

Victor-Marius Costan, București
Radu Berinde, București
Ștefan Ciobâcă, Suceava
Mircea Digulescu, București

Echipa s-a întors în țară cu trei medalii: una de aur (*Victor-Marius Costan*), una de argint (*Radu Berinde*) și una de bronz (*Ștefan Ciobâcă*).

În continuare vă prezentăm enunțurile celor șase probleme propuse spre rezolvare.

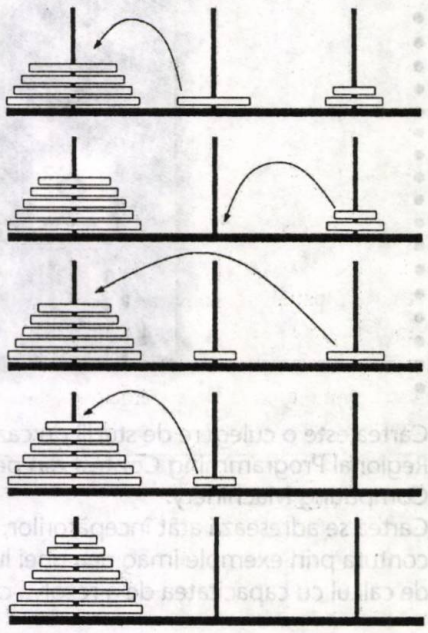
P060307: Hanoi

Toată lumea cunoaște problema *Turnurilor din Hanoi*: se consideră trei tije pe care sunt așezate discuri de mărimi diferite. Inițial toate se află pe aceeași tijă, aranjate în ordine descrescătoare de la bază spre vârf. Discurile trebuie mutate de pe tija sursă pe tija destinație, utilizând o tijă de manevră, respectând regula conform căreia pe un disc se poate așeza un altul doar dacă are diametrul mai mic.

Legenda povestește despre călugării dintr-o mănăstire tibetană care încearcă de mii de ani să rezolve problema, folosind 47 de discuri. Deoarece sunt necesare $2^{47} - 1$ mutări, nu au reușit să rezolve problema nici până în prezent. Acum discurile sunt împrăștiate pe trei tije, pe fiecare aflându-se un număr oarecare de discuri. Călugării doresc să rearanjeze discurile pe o singură tijă cu număr minim de mutări, respectând regulile privind mutările. Ei doresc să cunoască pe care tijă trebuie să mute discurile precum și numărul minim de mutări necesare.

Programul vostru trebuie să fie capabil să mute N discuri. Numerele reprezentând rezultatele parțiale, pot fi foarte mari. Din această cauză călugării sunt interesați de numărul mutărilor modulo 1.000.000.

În figura următoare este prezentată o configurație care poate fi rezolvată din patru mutări.



Date de intrare

Pe prima linie a fișierului de intrare **`hanoi.in`** este scris numărul N, reprezentând numărul de discuri.

Cea de-a doua linie conține trei numere întregi s_1, s_2, s_3, unde $0 \le s_1, s_2, s_3 \le N$ și $s_1 + s_2 + s_3 = N$, reprezentând numărul discurilor de pe fiecare dintre cele trei tije.

Liniile 3-5 conțin fiecare dimensiunile discurilor aflate pe fiecare dintre cele trei tije. Mai precis: cea de a $(i + 2)$-a linie din fișierul de intrare conține numerele întregi $m_{i,1}$, ..., $m_{i,si}$, unde $1 \le m_{i,j} \le N$ reprezintă mărimile discurilor de pe tija i. Diametrele discurilor sunt precizate de la bază spre vârf, adică $m_{i,1} > m_{i,2} > ... > m_{i,si}$. Rețineți că o tijă fără discuri este reprezentată printr-o linie vidă. Dimensiunile diametrelor celor N discuri formează o mulțime de numere distincte. Toate numerele sunt separate printr-un singur spațiu.

Date de ieșire

Prima linie a fișierului de ieșire **hanoi.out** va conține numărul de ordine al tijei ($d \in \{1, 2, 3\}$) pe care se vor așeza discurile printr-un număr minim de mutări. Pe cea de a doua linie se va scrie numărul M, reprezentând numărul de mutări cerut modulo 1 000 000.

Restricție

- $1 < N \le 100.000$.

Exemplu

hanoi.in

```
7
2 1 4
2 1
3
7 6 5 4
```

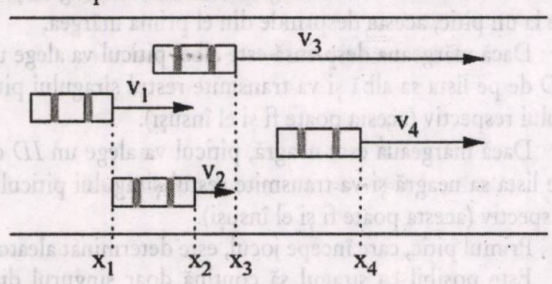
hanoi.out

```
3
4
```

Timp de execuție: 1 secundă/test

P060308: Cursa

În timpul *Competiției Anuale pentru Navete Spațiale* se întrec N vehicule spațiale. Fiecare navă spațială i este lansată astfel încât poate să accelereze instantaneu până la viteza maximă V_i, rămânând fixată pe această viteză. Fiecare navă pornește de la o poziție X_i, specificându-se câți kilometri o despart de linia de start.



Cursa are lungime infinită. Datorită faptului că vitezele navelor sunt foarte mari, traiectoriile navelor sunt liniare. Navele se pot depăși fără ca traiectoriile lor să se intersecteze (acestea sunt paralele).

Sarcina voastră este să determinați câte depășiri vor avea loc și să precizați primele 10.000 de depășiri în ordine cronologică.

Rețineți că navele pornesc din poziții diferite și că la un anumit moment nu vor exista mai mult de două nave în aceeași poziție în timpul cursei.

Date de intrare

Prima linie a fișierului **therace.in** conține numărul N al navelor participante la cursă. Pe fiecare dintre următoarele N linii sunt descrise proprietățile uneia dintre nave.

Pe cea de-a $(i + 1)$-a linie a fișierului se descrie a i-a navă prin două numere întregi X_i și V_i, reprezentând poziția de start și viteza navei respective. Navele sunt aliniate la start după poziție adică $X_1 < X_2 < ... < X_N$. Poziția de start a navei este reprezentată prin numărul de kilometri ce o despart de linia de start, iar viteza este măsurată în kilometri pe secundă.

Date de ieșire

Prima linie a fișierului de intrare **therace.out** va conține numărul de depășiri pe perioada cursei modulo 1.000.000.

Pe liniile următoare se vor descrie depășirile în ordine cronologică. În cazul în care sunt mai mult de 10.000 de depășiri, doar primele 10.000 se vor afișa. Dacă sunt mai puțin de 10.000, se vor afișa toate.

Fiecare linie va conține două numere întregi i și j, reprezentând faptul că nava i o depășește pe nava j.

Dacă există mai multe depășiri în același timp, acestea vor fi sortate după pozițiile lor din cursă, deci depășirile care au loc mai aproape de punctul de start se vor scrie mai repede.

Restricții și precizări

- $0 < N \le 250\ 000$;
- $0 \le X_i \le 1.000.000$, $i = 1, 2, ... N$;
- $0 < V_i < 100$, $i = 1, 2, ... N$.

Exemplu

therace.in

```
4
0 2
2 1
3 8
6 3
```

therace.out

```
2
3 4
1 2
```

Observație

Dacă găsiți numărul corect de depășiri veți primi 40% din scorul pe test.

Găsirea corectă a primelor 10.000 de depășiri va fi punctată cu 60% din scor. În cazul în care programul se termi-

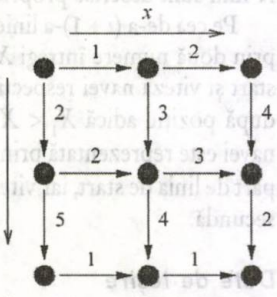
nă normal în cadrul timpului alocat, cele două părți din fișierul de ieșire se vor puncta separat.

Pentru această problemă sunt 20 de teste și se cunoaște conținutul fișierelor de test. Pentru fiecare test în parte valorile lui N sunt: 5, 10, 15, 20, 50, 100, 150, 200, 1000, 2000, 3000, 4000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, respectiv 100000.

Timp de execuție: 1 secundă/test

P060309: Pătrat

Se consideră un graf ale cărui noduri sunt punctele de intersecție ale unui caroiaj de dimensiune $N \times N$. Fiecare nod are muchii orientate spre nodurile vecine din dreapta și de jos, dacă un astfel de nod există. Muchiile au costuri având valori întregi w. Orice drum care pornește din nodul din stânga-sus (situat în poziția $v_{1,1}$) către nodul $v_{x,y}$ are același cost. Costul unui drum este egal cu suma costurilor muchiilor de-a lungul drumului respectiv.



În figura alăturată este prezentat graful corespunzător valorii $N = 3$. Fiecare cost de la nodul $v_{1,1}$ la $v_{2,2}$ este egal cu 4. Singurul drum de la $v_{1,1}$ la $v_{1,2}$ are costul 2.

Cunoscând valoarea numărului întreg L, determinați vârful $v_{x,y}$ pentru care costul drumului de la $v_{1,1}$ la $v_{x,y}$ este exact L.

Costurile nu sunt date direct programului, ci acesta le va cere prin intermediul unei biblioteci. Programul vostru va pune cel mult 6667 de întrebări referitoare la costuri.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul vostru, pentru limbajul *Pascal*, instrucțiunea `uses square_lib.h;`.

Dacă utilizați limbajul *C/C++*, va trebui să includeți în programul vostru, secvența: `#include "square_lib.h"`.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

```
function getN: Longint;
int getN()
```
• returnează valoarea lui N.

```
function getL: Longint;
int getL()
```
• returnează valoarea lui L.

```
function getWeight(x, y, directie: Longint):
                                        Longint;
int getWeight(int x, int y, int directie)
```
• întrerupe execuția programului dacă este apelată de mai mult de 6667 de ori, caz în care veți primi 0 puncte pentru testul respectiv;
• returnează costul drumului pornind din $v_{x,y}$ spre dreapta (`directie = 0`) sau în jos (`directie = 1`).

```
procedure solution(x, y: Longint);
void solution(int x, int y)
```
• se apelează în momentul în care ați găsit nodul $v_{x,y}$ pentru care costul drumului de la $v_{1,1}$ până la el este L;
• calculează punctajul și întrerupe execuția programului.

Restricții și precizări

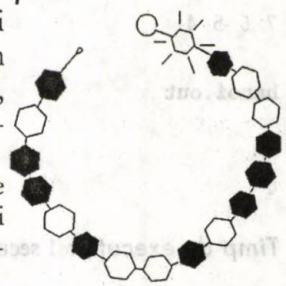
• $0 \le N \le 2003$;
• $0 \le w \le 500.000$;
• $0 \le L \le 2.000.000$;
• în cazul în care nu ați găsit nici o soluție, apelați funcția `solution` cu parametrii `-1` și `-1`(`solution(-1,-1)`).

Exemplu de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
`GetN()`	3
`getL()`	4
`getWeight(1,1,0)`	1
`getWeight(2,1,1)`	3
`solution(2,2)`	-

Timp de execuție: 0,1 secunde/test

P060310: Colierul de perle



Un grup de pitici roșii și verzi au găsit un șirag format din mărgele de sticlă negre și albe, la sfârșitul căruia se află un valoros diamant.

Fiecare clan dorește să intre în posesia diamantului. Piticii decid să rezolve problema amiabil, jucând următorul joc.

Cei N pitici sunt numerotați de la 1 la N, număr numit în continuare *ID*. Fiecărui pitic i se comunică două liste de *ID*-uri: o listă albă și una neagră. (Listele pot fi diferite de la un pitic la altul.) Fiecare dintre aceste liste conține *ID*-uri ale unor pitici roșii și verzi.

În timpul jocului șiragul se transmite de la un pitic la altul folosind următoarele reguli: atunci când șiragul ajunge la un pitic, acesta desprinde din el prima mărgea.

Dacă mărgeaua desprinsă este albă, piticul va alege un *ID* de pe lista sa albă și va transmite restul șiragului piticului respectiv (acesta poate fi și el însuși).

Dacă mărgeaua este neagră, piticul va alege un *ID* de pe lista sa neagră și va transmite restul șiragului piticului respectiv (acesta poate fi și el însuși).

Primul pitic, care începe jocul, este determinat aleator.

Este posibil ca șiragul să conțină doar singurul diamant. Jocul este câștigat de acel pitic care ajunge în posesia diamantului.

Sarcina voastră este să scrieți un program care să îi ajute pe piticii verzi să obțină diamantul.

Pentru a rezolva problema aveți la dispoziție o bibliotecă și un fișier de intrare și puteți presupune că piticii roșii joacă optim.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul vostru, pentru limbajul *Pascal*, instrucțiunea `uses pearls_lib;` și, pentru limbajul *C/C++*, `#include "pearls_lib.h"`.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

```
function getNext: Integer;
int getNext()
```

- trebuie apelată atunci când este rândul unui pitic roșu;
- returnează *ID*-ul piticului căruia piticul roșu îi va da restul șiragului;
- întrerupe execuția programului dacă este apelată când este rândul piticului verde, caz în care veți primi 0 puncte pentru testul respectiv.

```
procedure setNext(d: Integer);
void setNext(int d)
```

- trebuie apelată atunci când este rândul unui pitic verde;
- parametrul *d* specifică *ID*-ul piticului căruia piticul verde îi va da restul șiragului;
- întrerupe execuția programului dacă este apelată când este rândul piticului roșu sau dacă valoarea lui *d* nu este validă, caz în care veți primi 0 puncte pentru testul respectiv.

```
procedure finish;
void finish(void)
```

- se apelează la sfârșitul jocului când programul vostru se termină;
- calculează punctajul obținut și întrerupe execuția programului (în cazul în care piticul roșu este în posesia diamantului sau dacă șiragul conține mărgele pe lângă diamant veți primi 0 puncte pentru testul respectiv).

Date de intrare

Prima linie a fișierului de intrare **pearls.in** conține trei numere naturale: lungimea inițială *L* a șiragului, numărul *N* al piticilor și *F* ($1 \le F \le N$) care reprezintă *ID*-ul piticului care începe jocul. Rețineți că $1 \le i \le N$, pentru orice *i* reprezentând un *ID*.

A doua linie conține *L* caractere care descriu șiragul. Primele *L* - 1 caractere sunt litere *B* sau *W*. *B* reprezintă mărgeaua neagră, iar *W* reprezintă mărgeaua albă. Ultimul caracter este litera *D*, reprezentând diamantul.

Următoarele *N* linii descriu piticii împreună cu listele lor. A *i*-a linie descrie piticul având *ID*-ul *i*. Primul număr de pe o astfel de linie reprezintă culoarea piticului: 0 pentru un pitic verde, 1 pentru un pitic roșu. Urmează numărul L_B, reprezentând numărul piticilor din lista neagră, apoi cele L_B *ID*-uri din lista neagră a piticului respectiv ($1 \le L_B \le 20$). Urmează numărul L_W, reprezentând numărul piticilor din lista albă, apoi cele L_W *ID*-uri din lista albă a piticului respectiv ($1 \le L_W \le 20$).

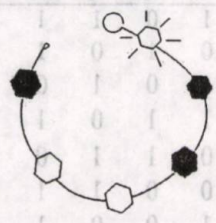
Restricții și precizări

- $1 \le L, N \le 1.000$;
- testele sunt construite astfel încât este posibil întotdeauna ca piticul verde să intre în posesia diamantului.

Exemplu de utilizare a bibliotecii

pearls.in

```
6 4 2
BWWBBD
0 1 2 1 4
0 2 1 3 1 1
1 1 4 1 4
1 2 2 3 1 1
```



Subprogram apelat	*Valoare returnată*
`setNext(1)`	-
`setNext(4)`	-
`getNext()`	1
`setNext(2)`	-
`setNext(1)`	-
`finish()`	-

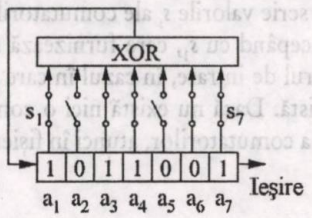
Timp de execuție: 2 secunde/test

P060311: Registru de deplasare

Un registru al unui calculator înmagazinează *N* biți. Biții memorați într-un registru de shiftare se translatează cu o poziție.

Folosind un registru de deplasare spre dreapta se pot genera numere binare aleatoare în felul următor: un registru de deplasare de dimensiune *N* se umple inițial cu biți având valorile $a_1, a_2, ..., a_N$.

La fiecare tact al ceasului sistemului de calcul, registrul furnizează la ieșire valoarea celui mai din dreapta bit a_N. Ceilalți biți sunt translatați cu o poziție spre dreapta. Prima poziție este asociată unei noi valori a'_1 astfel:



Fiecare bit din registru este conectat la o poartă *XOR* printr-un comutator (figura anterioară); adică pentru fiecare bit *i* există un comutator de valoare s_i (care este 1 sau 0) și care determină dacă valoarea bitului a_i este trimisă spre poarta *XOR* sau nu.

Fie $k_i = s_i \cdot a_i$. Noua valoare a'_1 este atribuită valorii de ieșire a porții *XOR*, $XOR(k_1, k_2, ..., k_N)$. (Notă: dacă numărul elementelor egale cu 1 din secvența $k_1, k_2, ..., k_N$, este impar, atunci valoarea lui $XOR(k_1, k_2, ..., k_N)$ este 1, altfel este 0).

Considerăm următoarele definiții formale:
- $a'_1 = XOR(k_1, k_2, ..., k_N)$;
- $a'_i = a_{i-1}$, pentru $2 \le i \le N$;
- *valoarea furnizată la ieșire* = a_N.

În exemplul din figura următoare valoarea a_1 la tactul 1 se calculează astfel:

$XOR(1 \cdot 1, 0 \cdot 0, 1 \cdot 1, 1 \cdot 1, 0 \cdot 0, 1 \cdot 0, 1 \cdot 1) = 0$.

tact	a_1	a_2	a_3	a_4	a_5	a_6	a_7	ieșire
0	1	0	1	1	0	0	1	-
1	0	1	0	1	1	0	0	1
2	1	0	1	0	1	1	0	0
3	1	1	0	1	0	1	1	0
4	0	1	1	0	1	0	1	1
5	0	0	1	1	0	1	0	1
6	1	0	0	1	1	0	1	0
7	1	1	0	0	1	1	0	1
8	0	1	1	0	0	1	1	0
9	1	0	1	1	0	0	1	1
10	0	1	0	1	1	0	0	1
11	1	0	1	0	1	1	0	0
12	1	1	0	1	0	1	1	0
13	0	1	1	0	1	0	1	1
14	0	0	1	1	0	1	0	1

Se consideră primele $2 \cdot N$ valori furnizate la ieșire ale unui astfel de registru de deplasare. Pe baza acestor valori trebuie să determinați valorile s_i ale comutatorilor.

Date de intrare

Prima linie a fișierului de intrare **register.in** conține dimensiunea N a registrului de deplasare. A doua linie conține $2 \cdot N$ numere de 0 sau 1, reprezentând primele $2 \cdot N$ valori ale biților furnizați la ieșire de către registrul de deplasare.

Date de ieșire

Fișierul de ieșire **register.out** va conține o singură linie pe care veți scrie valorile s_i ale comutatorilor dintr-o configurație, începând cu s_1, care furnizează secvența de biți dată în fișierul de intrare, în cazul în care o astfel de configurație există. Dacă nu există nici o configurație corespunzătoare a comutatorilor, atunci în fișierul de ieșire veți scrie -1.

Restricție

- $1 \le N \le 750$.

Exemple

register.in
```
7
1 0 0 1 1 0 1 0 1 1 0 0 1 1
```

register.out
```
1 0 1 1 0 1 1
```

register.in
```
3
0 0 0 1 1 1
```

register.out
```
-1
```

Timp de execuție: 1,5 secunde/test

P060312: Călătoria

Alice și *Bob* vor să meargă în vacanță. Cei doi și-au planificat câte un traseu pe care doresc să-l urmeze și au scris o listă de orașe pe care vor să le viziteze într-o anumită ordine. O listă poate să conțină un anumit oraș de mai multe ori.

Deoarece cei doi vor să călătorească împreună, trebuie să cadă de acord privind un traseu comun. Nici unul nu vrea să schimbe ordinea în care au planificat vizitarea orașelor conform listei proprii și nu vor să adauge orașe noi la ea. În concluzie, ei nu au altă posibilitate decât să șteargă anumite orașe din lista proprie. Bineînțeles, traseul comun trebuie să conțină cel mai mare număr de orașe posibile de vizitat.

În regiune există exact 26 de orașe, care au fost codificate cu litere mici de la `'a'` la `'z'`.

Date de intrare

Fișierul de intrare **trip.in** conține două linii; prima linie conține lista lui *Alice*, cea de a doua, lista lui *Bob*. Fiecare listă conține cel puțin 1 caracter literă mică și cel mult 80 de caractere litere mici, care nu sunt separate prin nici un spațiu.

Date de ieșire

Fișierul de ieșire **trip.out** va trebui să conțină toate traseele care respectă condițiile descrise mai sus și fiecare traseu se va scrie în fișier o singură dată. Pe o linie se va scrie un singur traseu.

Restricții și precizări

- se garantează existența a cel puțin unui traseu nevid și cel mult 1000 de trasee diferite;
- traseele pot fi scrise în fișierul de ieșire în orice ordine.

Exemplu

trip.in
```
abcabcaa
acbacba
```

trip.out
```
ababa
abaca
abcba
acbca
acaba
acaca
acbaa
```

Timp de execuție: 7 secunde/test

Iași, România

Olimpiada BALCANICĂ de informatică

În perioada 14-20 iulie 2003 a avut loc cea de-a XI-a ediție a Olimpiadei Balcanice de Informatică. Acest concurs a fost inițiat de țara noastră în anul 1993. După zece ani Balcaniada s-a întors în România, ediția din acest an a concursului desfășurându-se la Iași.

Beneficiind de avantajele țării organizatoare, România a avut dreptul de a avea două echipe în concurs. Cei opt concurenți din România care au participat la această competiție sunt *Leonard Crestez*, *Dan Ghinea*, *Emilian Miron*, *Marius Nicolae*, *Alexandru Moșoi*, *Matei Gruber*, *Tiberiu Dăneț* și *Dan-Ionuț Fechete*.

Primii doi au obținut medalii de aur, următorii doi medalii de argint, iar ceilalți patru medalii de bronz.

În continuare vă prezentăm enunțurile celor șase probleme propuse spre rezolvare.

P060313: Secvența Farey

Secvența *Farey* de ordin N este definită ca fiind o secvență ordonată crescător care conține toate fracțiile ireductibile de forma p/q pentru care $0 < p < q \leq N$. De exemplu, secvența *Farey* de ordin 5 este următoarea:

```
1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5.
```

Numerotăm toate elementele secvenței începând cu valoarea 1. De exemplu, fracția cu numărul de ordin șase în secvența anterioară este `3/5`.

Dându-se două valori N și K, determinați fracția cu numărul de ordine K în secvența *Farey* de ordin N.

Date de intrare

Fișierul de intrare **`farey.in`** conține o singură linie pe care se află valorile N și K, separate printr-un spațiu.

Date de ieșire

Fișierul de ieșire **`farey.out`** va conține o singură linie pe care se vor afla numerele întregi p și q, separate printr-un spațiu. Fracția p/q trebuie să aibă numărul de ordine K în secvența *Farey* de ordin N.

Restricții și precizări

- $1 \leq N \leq 40000$;
- pentru 40% dintre seturile de date folosite pentru testare vom avea $K \leq 40000$;
- pentru toate seturile de date folosite pentru testare vom avea $N \geq 10000$;
- valoarea K va reprezenta întotdeauna un număr de ordine valid; cu alte cuvinte, va fi mai mare sau egală cu 1 și mai mică sau egală cu numărul elementelor secvenței *Farey* de ordin N;
- fracția p/q trebuie să fie ireductibilă; cu alte cuvinte, cel mai mare divizor comun al numerelor scrise în fișierul de ieșire trebuie să fie 1.

Exemplu

`farey.in`

```
5 6
```

`farey.out`

```
3 5
```

Timp de execuție: 0,3 secunde/test

P060314: Traversare

Am ajuns la un lac mare și trebuie să îl traversez. Din fericire, sunt un săritor foarte bun, dar lacul este mult prea mare pentru ca să îl pot traversa dintr-o singură săritură. Pe mal găsesc N cutii de diferite înălțimi, amplasate într-o anumită ordine. Dacă arunc o cutie în lac, ea va pluti și va avea înălțimea pe care o are pe mal. Acesta este un lucru bun, deoarece intenționez să arunc câteva cutii în lac și să ajung de la un mal la altul sărind de pe o cutie pe alta.

Trebuie să iau în considerare următoarele:

- lacul este mare, deci trebuie să arunc cel puțin două cutii, ceea ce înseamnă că pentru a traversa lacul, va trebui să sar de cel puțin trei ori;

- nu trebuie să arunc toate cutiile; pot să ignor unele dintre ele;
- pot să arunc cutiile în lac doar în ordinea în care acestea se află pe mal; așadar, trebuie să sar de pe una pe alta în această ordine;
- pot sări de pe o cutie pe alta doar dacă diferența dintre înălțimile celor două cutii este de cel mult H metri deoarece pot să sar destul de mult în lungime, dar am unele probleme cu săriturile în înălțime;
- înălțimea unei cutii nu se modifică în momentul în care sar pe ea;
- pot întotdeauna să sar de pe mal pe o cutie și de pe o cutie pe mal indiferent care este înălțimea cutiei respective;

Confruntându-mă cu numărul mare de posibilități care respectă condițiile descrise anterior, am început să număr variantele în loc să traversez lacul. Am găsit răspunsul foarte repede și mă întreb dacă îl puteți și voi găsi la fel de repede.

Scrieți un program care determină numărul posibilităților de a traversa lacul respectând condițiile anterioare. Datorită faptului că numărul poate fi foarte mare, va trebui să determinați doar restul împărțirii sale la 9901.

Date de intrare

Prima linie a fișierului de intrare **travers.in** conține numerele întregi N și H, separate printr-un spațiu. Aceste numere reprezintă numărul cutiilor, respectiv diferența maximă a înălțimilor a două cutii consecutive.

Următoarele N linii conțin câte un număr reprezentând înălțimea unei cutii. Înălțimile sunt date în ordinea în care se află cutiile pe mal. Cea de-a $(i + 1)$-a linie a fișierului va conține înălțimea celei de-a i-a cutii.

Date de ieșire

Fișierul de ieșire **travers.out** va conține o singură linie pe care se va afla un singur număr, reprezentând restul împărțirii numărului de posibilități la 9901.

Restricții și precizări

- $2 \le N \le 100000$;
- $1 \le H \le 100000000$;
- înălțimile cutiilor sunt numere întregi strict pozitive, cel mult egale cu 100000000;
- pentru 40% dintre seturile de date folosite pentru testare, înălțimile cutiilor vor fi numere întregi cel mult egale cu 100000.

Exemplu

travers.in	travers.out
4 2 1 3 7 5	4

Explicație

Cele patru posibilități sunt:

1 3
1 3 5
3 5
7 5

Timp de execuție: 0,5 secunde/test

P060315: Compania

O companie internațională are N angajați, numerotați de la 1 la N. Aceștia sunt organizați într-o structură ierarhică cu următoarele proprietăți:

- fiecare angajat are un superior direct, cu excepția angajatului cu numărul 1 care este managerul companiei;
- fiecare angajat poate fi superiorul direct a 0, 1 sau cel mult 2 alți angajați.

Pentru a îmbunătăți eficiența afacerii lor, deținătorii companiei au decis să o împartă în trei companii mai mic. Pe lângă alte probleme pe care le-au avut de rezolvat a apărut una foarte importantă: cum trebuie distribuiți angajații la cele trei companii nou create? Trebuie luate în considerare următoarele condiții:

- fiecare angajat trebuie distribuit la una dintre cele trei companii;
- un angajat și superiorul său nu trebuie să fie distribuiți la aceeași companie;
- dacă doi angajați au același superior, atunci ei trebuie să fie distribuiți la companii diferite.

Pentru ca cele trei companii să fie la fel de eficiente a mai apărut o condiție suplimentară. Să presupunem că N_1 este numărul de angajați distribuiți la prima companie, N_2 este numărul de angajați distribuiți la cea de-a doua companie și N_3 este numărul de angajați distribuiți la cea de-a treia companie ($N_1 + N_2 + N_3 = N$). Cu aceste notații, trebuie ca diferența $\max(N_1, N_2, N_3) - \min(N_1, N_2, N_3)$ să fie cât mai mică posibil.

Date de intrare

Fișierul de intrare **comp.in** conține pe prima linie un singur număr natural N, care reprezintă numărul de angajați ai companiei.

Fiecare dintre următoarele $N - 1$ linii conține două numere naturale a și b cuprinse între 1 și N, separate printr-un singur spațiu, care semnifică faptul că angajatul b este superiorul direct al angajatului a.

Date de ieșire

Fișierul de ieșire **comp.out** va avea o singură linie pe care se vor afla N numere, separate prin spații, din mulțimea {1, 2, 3}.

Primul număr reprezintă numărul companiei la care a fost distribuit angajatul cu numărul 1, al doilea număr reprezintă numărul companiei la care a fost distribuit an-

gajatul cu numărul 2, al treilea număr reprezintă numărul companiei la care a fost distribuit angajatul cu numărul 3 etc.

Restricții și precizări

- $0 \le N \le 16.000$;
- $\max(a, b, c)$ reprezintă maximul dintre valorile a, b și c;
- $\min(a, b, c)$ reprezintă minimul dintre valorile a, b și c;
- este acceptată orice soluție care minimizează diferența specificată în enunț;
- datele de intrare sunt corecte.

Exemplu

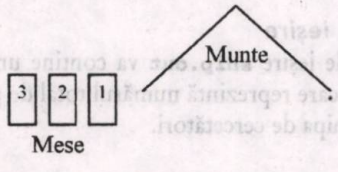
comp.in	**comp.out**
6 2 1 3 2 4 2 5 3 6 3	3 1 3 2 1 2

Timp de execuție: 0,1 secunde/test

P060316: *Consiliu de trib*

Tribul *Kazooba* are în componența sa N bărbați care vor să se întâlnească pentru un consiliu. Toți bărbații sunt descendenții unei singure persoane care este șeful tribului. De când tribul s-a format, toți strămoșii bărbaților sunt în viață.

Bărbații vin la consiliu într-o anumită ordine și se așează la mese în momentul în care ajung. Bărbații stau într-un rând de mese paralele. Prima masă se află la baza *Muntelui Sfânt* și rândul se extinde nedefinit către vest (mai departe de munte). Din păcate, nici un bărbat nu stă la aceeași masă cu fiii sau tatăl lui. Pe de altă parte, fiecare bărbat dorește să stea cât mai aproape posibil de divinitate, astfel, când un bărbat sosește la consiliu el se așează la masa cea mai apropiată de *Muntele Sfânt*, la care nu s-a așezat nici un fiu și nici tatăl său.



După cum se obișnuiește, zeii tribului au anumite înclinații pentru grandoare, așa că aceștia comandă ca bărbații să vină într-o ordine care să maximizeze numărul de mese ocupate. Din păcate, zeii nu au înclinații spre programare și v-au angajat să scrieți un program pentru ei.

Sarcina voastră este de a scrie un program care să determine numărul maxim de mese care pot fi ocupate după sosirea tuturor bărbaților care fac parte din trib.

Date de intrare

Prima linie a fișierului de intrare **tribe.in** conține un singur număr natural N, care reprezintă numărul bărbaților din tribul *Kazooba*. Fiecare bărbat este numerotat de la 1 la N, șeful tribului având numărul 1.

Fiecare dintre următoarele linii conține două numere naturale A și B cuprinse între 1 și N, separate printr-un singur spațiu, cu semnificația că bărbatul cu numărul de ordine A este tatăl bărbatului cu numărul B.

Date de ieșire

Fișierul de ieșire **tribe.out** va trebui să conțină un singur număr natural, care reprezintă numărul maxim de mese care sunt ocupate după sosirea la consiliu a tuturor bărbaților care fac parte din trib.

Restricții

- $1 < N \le 100.000$;
- pentru 30% din teste $N \le 1.000$;
- fiecare masă are capacitate infinită;
- datele de intrare sunt corecte.

Exemplu

tribe.in	**tribe.out**
5 1 4 3 2 1 3 3 5	3

Explicație

Dacă bărbații vin în ordinea 2 4 1 5 3, atunci:

- bărbatul cu numărul 2 se așează la prima masă;
- bărbatul cu numărul 4 se așează la prima masă;
- bărbatul cu numărul 1 are un fiu (4) la prima masă, deci acesta se așează la cea de-a doua masă;
- bărbatul cu numărul 5 se așează la prima masă;
- bărbatul cu numărul 3 are doi fii (2 și 5) la prima masă și a doua masă, deci acesta se așează la cea de-a treia masă.

Timp de execuție: 0,5 secunde/test

P060317: *Euro*

Tocmai v-ați deschis un cont la o bancă de unde veți primi banii în *euro*. Când v-ați deschis contul nu aveați nici *euro* în cont și nici *lei* în posesie. În fiecare dintre următoarele N zile vei primi sume în *euro*; acestea pot fi și negative, caz în care suma totală din contul tău va scădea. Este, de asemenea, posibil ca la un moment dat suma din cont să fie negativă.

La sfârșitul fiecărei zile, banca vă permite să transformați toată suma din cont, din *euro* în *lei*. Cursul de schimb se modifică zilnic după următoarea regulă: în timpul celei de-a K-a zi, un *euro* poate fi schimbat pentru K *lei*. Banca ia de asemenea T *lei* pentru fiecare conversie. Astfel, dacă la sfârșitul zilei aveți S euro în cont și vă decideți să schimbați toată suma în *lei* atunci veți primi $S \cdot K - T$ *lei* (S poate fi și negativ). La sfârșitul celei de-a N-a zi, trebuie să schimbați toată suma din *euro* în *lei*, chiar dacă este negativă.

Scopul vostru este acela de a maximiza suma totală în *lei* pe care o aveți la sfârșitul perioadei de N zile.

Date de intrare

Prima linie a fișierului de intrare **euro.in** conține două numere întregi N și T, separate printr-un singur spațiu, care reprezintă numărul de zile, respectiv suma pe care banca o ia la fiecare conversie.

Următoarea linie conține N numere întregi, separate prin spații, care reprezintă sumele în *euro* pe care le primiți la începutul fiecărei zile.

Date de ieșire

Pe prima linie a fișierului de ieșire **euro.out** va trebui să scrieți un singur număr întreg, care reprezintă suma maximă în *lei* pe care o puteți obține.

Restricții și precizări

- $1 \le N \le 34.567$;
- $1 \le T \le 34.567$;
- pentru 80% din teste, T va fi mai mic sau egal cu 255;
- suma în *euro* pe care o veți primi în fiecare zi este un număr întreg cuprins între -1.000 și 1.000;
- puteți converti doar *euro* în *lei*, nu și invers;
- pentru rezultat se recomandă folosirea tipurilor `int64` sau `extended` în limbajul *Pascal* și `long long` sau `double` în limbajul *C/C++* (în cazul în care efectuați calculele folosind numere întregi sau reale).

Exemplu

euro.in	**euro.out**
7 1 -10 3 -2 4 -6 2 3	17

Explicație

La sfârșitul primei zile schimbați -10 *euro* în *lei* și veți primi - 11 = - 10 · 1 - 1 *lei*.

A doua conversie are loc la sfârșitul celei de cincia zi, când aveți -1 *euro* în cont (3 - 2 + 4 - 6) și veți primi -6 = -1 · 5 - 1 *lei*.

Ultima conversie are loc la sfârșitul celei de șaptea zi când aveți 5 *euro* în cont (2 + 3) și veți primi pentru aceștia 34 = 5 · 7 - 1 *lei*.

Suma finală pe care o aveți în lei este 17 = - 11 - 6 + 34.

Timp de execuție: 0,4 secunde/test

P060318: Nava spațială

Asociația Spațială Română (*Romanian Space Association - RSA*) a descoperit un asteroid nou pe care l-a numit *BOI2003*. *RSA* este nerăbdătoare să exploreze asteroidul și, în acest scop a construit o navă spațială care să-i transporte pe cercetători pe acesta. Nava spațială are câteva caracteristici neobișnuite. În momentul în care nava spațială părăsește pământul pentru prima dată ar trebui să transporte exact S oameni de știință. Când nava spațială ajunge pe asteroidul *BOI2003*, un singur cercetător aterizează și nava se întoarce pentru a mai aduce S cercetători.

Primul cercetător care a ajuns pe *BOI2003*, suspectează existența unui virus periculos pe acesta. Datorită acestui fapt, el sugerează ca nici un cercetător care a călătorit pe *BOI2003* să nu părăsească nava spațială când ajunge pe pământ, până după terminarea cercetărilor.

Pentru a fi mai exacți, să presupunem că cercetătorii sunt numerotați începând cu 1. Nava spațială funcționează după cum urmează:

- prima dată când părăsește pământul, nava transportă cercetătorii 1, 2, ..., S; unul dintre aceștia coboară pe asteroid, iar ceilalți S - 1 se întorc pe pământ, fără însă a părăsi nava;
- a doua oară părăsește pământul, după ce cercetătorii $S + 1$, $S + 2$, ..., $2 \cdot S$ se alătură celor S - 1 cercetători care s-au întors, deci nava transportă $2 \cdot S$ - 1 cercetători; unul dintre aceștia coboară pe asteroid, iar ceilalți $2 \cdot S$ - 2 se întorc pe pământ fără însă a părăsi nava;
- a treia oară, cercetătorii $2 \cdot S + 1$, $2 \cdot S + 2$, ..., $3 \cdot S$ împreună cu cei $2 \cdot S$ - 2 cercetători care s-au întors sunt transportați la asteroid și, din nou, doar unul dintre aceștia coboară, iar ceilalți $3 \cdot S$ - 3 se întorc pe pământ fără însă a părăsi nava;
- ș.a.m.d

După N transporturi, o echipă formată din N cercetători care au fost transportați se află pe asteroid și îl cercetează.

Sarcina voastră este de a scrie un program care să determine numărul total de posibilități de a forma echipa de cercetători.

Date de intrare

Fișierul de intrare **ship.in** conține o singură linie pe care se află două numere întregi S și N, separate printr-un singur spațiu, care reprezintă capacitatea navei spațiale, respectiv numărul de transporturi care se vor efectua.

Date de ieșire

Fișierul de ieșire **ship.out** va conține un singur număr întreg K, care reprezintă numărul total de posibilități de a forma echipa de cercetători.

Restricții și precizări

- $1 \le S \le 10$;
- $1 \le N \le 40$;
- capacitatea navei spațiale este nelimitată;
- ordinea în care membrii echipei ajung la asteroid nu este importantă.

Exemplu

ship.in	**ship.out**
2 3	14

Timp de execuție: 0,4 secunde/test

Satu Mare, 29 martie 2003

Concursul INTERJUDEȚEAN de programare GRIGORE MOISIL

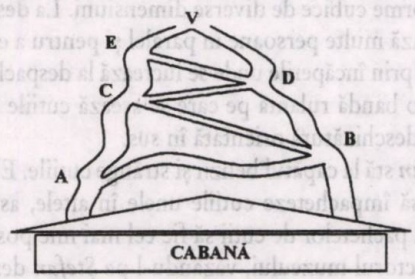
Ediția din acest an a concursului interjudețean de programare Grigore Moisil a avut loc la Satu Mare pe data de 29 martie 2003. În continuare vă prezentăm enunțurile celor șapte probleme propuse spre rezolvare la acest concurs.

Clasa a IX-a

P060319: *Excursie*

Pe un vârf înalt din Munții Retezat se poate urca pe două poteci abrupte care pornesc de la cabană. Cele două poteci sunt legate din când în când printr-o serpentină, adică o potecă mai comodă care leagă două puncte oarecare de pe cele două poteci abrupte, astfel încât pe aceste porțiuni se poate urca mai domol.

Un grup de turiști ar vrea să știe în câte feluri ar putea să urce spre vârf, folosind toate modalitățile posibile de a urca pe porțiuni de potecă abruptă, apoi pe porțiuni de serpentine, iar pe potecă abruptă etc.



Să se determine numărul tuturor traseelor pe care turiștii ar putea să urce spre vârf, știind că ei vor merge întotdeauna în sus spre vârf. De exemplu, corespunzător figurii sunt posibile porțiunile *cabană* → *A*, *A* → *C*, sau *A* → *B*, apoi *cabană* → *B* etc., dar nu sunt posibile *C* → *A*, *B* → *A* etc.

Date de intrare

În fișierul de intrare **EXCURSIE.IN** se află un număr natural *n*, reprezentând numărul total de intersecții ale potecilor abrupte și ale serpentinelor.

Date de ieșire

În fișierul de ieșire **EXCURSIE.OUT** se va scrie un singur număr, reprezentând numărul traseelor diferite dintre punctul de pornire și vârf.

Restricție

- $1 \le n \le 40$.

Exemplu

EXCURSIE.IN
```
5
```

EXCURSIE.OUT
```
13
```

Timp de execuție: 1 secundă/test

P060320: *Loto*

Vasile ar vrea cu orice preț să câștige la *loto*. El își notează toate numerele posibile de jucat, apoi scrie un program care să-i genereze configurații de numere care ar putea să fie trecute pe biletele de *loto*. Programul lui generează toate variantele posibile (combinații de 6 numere din 49) și de asemenea, numerotează variantele începând cu numărul de ordine 1. *Vasile* și-a dat seama că numărul variantelor fiind foarte mare, nu are nici o șansă să cumpere atâtea bilete câte variante sunt. În concluzie, el hotărăște să aleagă prima variantă, o trece pe un bilet, apoi selectează fiecare a *k*-a variantă din lista generată de program.

Atunci când ajunge la capătul listei, continuă numărătoarea de la începutul listei, ca și cum aceasta ar fi circulară. El termină selectarea variantelor atunci când, datorită nu-

mărătorii efectuate, nimerește o variantă pe care a ales-o deja.

Vasile totuși este supărat și își imaginează un joc de *loto* în care se extrag cel mult y numere din totalul de x.

Ajutați-l pe *Vasile* și scrieți un program care calculează numărul total n al variantelor pe baza numerelor x și y.

Scrieți un alt program, care pe baza numerelor n și k date, determină numărul variantelor pe care *Vasile* nu le va juca dintre cele posibile, dacă el va aplica strategia descrisă.

Date de intrare

Pe prima linie a fișierului de intrare **LOTO1.IN** sunt scrise numerele naturale nenule x și y, despărțite printr-un spațiu.

Pe prima linie a fișierului de intrare **LOTO2.IN** se află două numere naturale nenule n și k, despărțite printr-un spațiu, reprezentând numărul de variante și numărul k ales de *Vasile*.

Date de ieșire

În fișierului de ieșire **LOTO1.OUT** veți scrie numărul variantelor calculat pe baza numerelor x și y, citite din fișierul de intrare.

În fișierul de ieșire **LOTO2.OUT** veți scrie numărul variantelor pe care nu le va juca *Vasile*.

Restricții și precizări

- două variante care conțin aceleași numere, dar în ordine diferită, se consideră identice;
- $1 < n \leq 13983816$;
- $1 < k < n$;
- $1 < x \leq 49$;
- $1 < y \leq 6$.

Exemple

LOTO1.IN
```
4 2
```

LOTO1.OUT
```
6
```

LOTO2.IN
```
6 2
```

LOTO2.OUT
```
3
```

LOTO1.IN
```
30 4
```

LOTO1.OUT
```
27405
```

LOTO2.IN
```
27405 5
```

LOTO2.OUT
```
21924
```

Timp de execuție: 1 secundă/test

Clasa a X-a

P060321: Împărțeală

Ancuța a primit de ziua ei n bomboane și ar vrea să le împartă printre prietenii cei mai apropiați. În scopul unei împărțeli speciale, ea a notat pe o hârtie numele celor k prieteni în ordinea importanței lor în viața ei și ar vrea ca:

- fiecare prieten să primească cel puțin o bomboană, și
- un prieten nu poate primi mai multe bomboane decât un altul care are numărul de ordine mai mic.

Observând cât de multe posibilități are, *Ancuța* s-a zăpăcit, dar totuși ar vrea să știe din câte posibilități va trebui să aleagă. Ajutați-o, scriind un program care calculează numărul posibilităților de a împărți cele n bomboane printre cei k prieteni, respectând cele două restricții stabilite de *Ancuța*.

Date de intrare

Fișierul de intrare **ANCA.IN** conține două numere naturale, despărțite printr-un spațiu, reprezentând numărul bomboanelor și numărul prietenilor.

Date de ieșire

În fișierul de ieșire **ANCA.OUT** se va scrie un număr natural care reprezintă numărul posibilităților pe care le are *Ancuța* la dispoziție pentru a împărți bomboanele printre prietenii săi.

Restricții

- $1 \leq n \leq 100$;
- $1 \leq k \leq 12$;
- trebuie împărțite toate bomboanele.

Exemplu

ANCA.IN
```
10 3
```

ANCA.OUT
```
8
```

Timp de execuție: 1 secundă/test

P060322: Cutii

S-a mutat muzeul. Obiectele au fost împachetate în cutii având forme cubice de diverse dimensiuni. La despachetare lucrează multe persoane în paralel și pentru a evita dezordinea, prin încăperile unde se lucrează la despachetare s-a instalat o bandă rulantă pe care se așează cutiile goale, cu singura deschizătură orientată în sus.

Ștefan stă la capătul benzii și strânge cutiile. El a primit sarcina să împacheteze cutiile unele în altele, astfel încât numărul pachetelor de cutii să fie cel mai mic posibil.

Directorul muzeului, văzându-l pe *Ștefan* dezorientat datorită acestei cerințe, s-a hotărât să-l ajute și a stabilit regulile următoare:

- cutiile se culeg de pe bandă în ordinea sosirii lor;
- cutia curentă se așează într-o altă cutie, dacă aceasta are dimensiunea mai mică;
- dacă nu există pachet început în care să încapă cutia curentă, aceasta va constitui prima cutie dintr-un pachet nou;
- într-un pachet început se așează o singură cutie (nu se pun mai multe cutii unele lângă altele, chiar dacă există loc);
- o cutie așezată la un moment dat nu se mai scoate;

- un pachet început nu se așează în alt pachet, chiar dacă acest lucru ar fi posibil;
- nici o cutie nu poate fi ignorată.

Scrieți un program care determină numărul minim de pachete de cutii rezultate în urma muncii lui *Ștefan*, precum și secvențele de cutii din fiecare pachet.

Date de intrare

Prima linie a fișierului de intrare **CUTII.IN** conține un singur număr natural n, reprezentând numărul cutiilor. Următoarele n linii conțin fiecare câte un număr natural, reprezentând dimensiunile cutiilor.

Date de ieșire

Pe prima linie a fișierului **CUTII.OUT** se va scrie numărul minim m de pachete de cutii. Pe următoarele m linii se vor scrie dimensiunile cutiilor pe care *Ștefan* le va împacheta într-un singur pachet, în ordinea împachetării.

Restricții

- $0 \le n \le 1000$;
- $1 \le dimensiune\_cutie \le 10.000$.

Exemplu

CUTII.IN

```
10
4
1
5
10
7
9
2
8
3
2
```

CUTII.OUT

```
4
4 1
5 2
10 7 3 2
9 8
```

Timp de execuție: 1 secundă/test

Clasele a XI-a și a XII-a

P060323: *Ali Baba și cei 40 de hoți*

Ali Baba și-a ascuns mărgelele adunate în boluri într-un loc numai de el știut. Din păcate un hoț l-a pândit și i-a condus pe prieteni la locul respectiv. Hoții, având un strop de conștiință, au decis să nu fure toate mărgelele ci doar o parte mică, în așa fel încât *Ali Baba* să nu observe nimic. Hoții au hotărât să confecționeze un șirag din acele tipuri de mărgele din care există cele mai multe. Au decis să verifice toate bolurile și să aleagă câte k mărgele identice (având aceeași culoare) din fiecare bol, dacă mai mult decât jumătate din mărgele au acea culoare și dacă există cel puțin k mărgele de aceeași culoare în bolul respectiv.

Hoții au lucrat la numărătoare, unii au reușit să pună deoparte mărgele, alții au ajuns la concluzia că nu pot lua mărgele din bolul pe care l-au verificat. În final, văzând că au reușit să adune destul de multe mărgele, unul dintre ei a propus să confecționeze șiragul doar din mărgele având aceeași culoare, dar și de data aceasta numai în situația în care, dintre mărgelele alese, mai mult decât jumătate au aceeași culoare.

Cunoscând numărul bolurilor și culorile mărgelelor din fiecare bol, stabiliți dacă hoții au putut sau nu să confecționeze șiragul de mărgele conform regulilor stabilite.

Date de intrare

Pe prima linie a fișierului de intrare **MARGELE.IN** s-a scris numărul n de boluri și numărul k reprezentând numărul de mărgele luate din fiecare bol în scopul confecționării șiragului de mărgele. Pe următoarele n linii sunt scrise culorile mărgelelor, despărțite printr-un spațiu (pe linia i sunt culorile mărgelelor din bolul i).

Date de ieșire

În cazul în care hoții vor reuși să confecționeze șiragul, pe prima linie a fișierului de ieșire **MARGELE.OUT** veți scrie culoarea șiragului, în caz contrar veți scrie NU. În cazul în care în fișier nu s-a scris NU, pe cea de a doua linie veți scrie numărul mărgelelor din șirag.

Restricții și precizări

- $1 < k \le 500$;
- numărul de boluri este cel mult 40;
- într-un bol toate mărgelele pot fi de culori diferite;
- într-un bol pot fi cel mult 2000 de mărgele;
- culoarea este specificată printr-o succesiune de cel mult 10 litere ale alfabetului latin.

Exemplu

MARGELE.IN

```
5 3
rosu rosu verde rosu
verde galben galben galben
albastru rosu alb
rosu rosu rosu albastru rosu
rosu albastru rosu
```

MARGELE.OUT

```
rosu
6
```

Explicație

Din primul bol se pot lua 3 mărgele roșii, din al doilea bol se pot lua 3 mărgele galbene, din cel de-al treilea bol nu se

pot lua 3 mărgele de culoare identică, din bolul 4 se vor lua 3 mărgele roșii, din bolul 5 nu se pot lua 3 mărgele de culoare identică. În total sunt 6 mărgele roșii și 3 galbene, deci șiragul va fi confecționat din 6 mărgele roșii.

Timp de execuție: 1 secundă/test

P060324: *Profi și studenți*

În universități, fiecare cadru didactic (cu excepția rectorului) are un singur șef direct și poate avea subalterni. Se știe că dacă rectorul nu este șeful direct al unui cadru didactic, atunci el este sigur șeful indirect al acestuia.

La sfârșitul ședințelor profesorii universitari mai zăbovesc un pic în sala de conferințe, apoi părăsesc sala unul câte unul. Studenții au observat că profesorii părăsesc sala pe baza unor reguli:

- nici o persoană nu părăsește sala atâta timp cât în sală mai există persoane care le sunt subordonate;
- dacă la un moment dat ar fi posibil să iasă din sală mai multe persoane, atunci va ieși cea care are numele (numărul de ordine în tabelul nominal) mai mic;
- fiecare persoană își salută șeful direct în momentul în care părăsește sala.

La mai multe universități studenții au numerotat profesorii pe baza ordinii alfabetice și au notat numerele de ordine ale cadrelor didactice salutate (este posibil ca un același număr să fie notat de mai multe ori) și vor să reconstituie ordinea în care profesorii au părăsit sala.

Date de intrare

Pe prima linie a fișierului **PROFI.IN** este scris numărul m al universităților. Următoarele $2 \cdot m$ linii conțin datele referitoare la cele m universități. Pe prima linie din a i-a pereche se află scris numărul n al cadrelor didactice din a i-a universitate. Pe cea de a doua linie din fiecare pereche de linii sunt scrise numerele de ordine ale profesorilor în ordinea în care au fost salutați.

Date de ieșire

Fișierul de ieșire **PROFI.OUT** va conține m linii. Răspunsul referitor la o universitate va conține un șir de numere naturale, despărțite printr-un spațiu, reprezentând numerele de ordine ale cadrelor didactice în ordinea în care au părăsit sala.

Restricții și precizări

- $1 \le m \le 100$;
- $2 \le n \le 10.000$, pentru orice universitate.

Exemplu

PROFI.IN

```
3
5
2 1 4 2
6
1 1 1 1 1
5
2 2 5 2
```

PROFI.OUT

```
3 5 1 4 2
2 3 4 5 6 1
1 3 4 5 2
```

Timp de execuție: 1 secundă/test

P060325: *Hacker*

Un *hacker* a găsit o matrice de caractere care este folosită de *FBI* la criptarea unor informații secrete. Pentru a putea decripta o informație importantă, el are nevoie de o cheie, care este un șir de caractere și de numărul de apariții al acestuia în matrice. Un șir de caractere apare în matrice dacă pozițiile sale se parcurg începând de la o poziție (i, j) fie la dreapta, fie la stânga, respectiv fie în sus, fie în jos, cât timp este nevoie pentru a regăsi fiecare caracter a cheii în ordinea elementelor aflate în matrice.

Membrii comisiei au aflat cheile de criptare și lucrează cu *hackerul* pentru spargerea datelor *FBI*-ului. Ajutați comisia să rezolve problema, determinând numărul de apariții a cheii.

Date de intrare

Pe prima linie a fișierului **HACKER.IN** este scris numărul n, reprezentând dimensiunea matricei. Următoarea linie conține șirul de caractere care reprezintă cheia de criptare. Următoarele n linii conțin câte n caractere, corespunzătoare matricei *FBI*-ului.

Date de ieșire

Fișierul de ieșire **HACKER.OUT** va conține pe prima linie numărul de apariții ale cheii în matricea dată la intrare.

Restricții și precizări

- $2 \le n \le 100$;
- în cheie caracterele sunt distincte;
- caracterele în cheie și în matrice sunt litere mari și mici ale alfabetului englezesc.

Exemplu

HACKER.IN

```
4
BCD
AABA
ABCD
AADA
AAAA
```

HACKER.OUT

```
4
```

Timp de execuție: 1 secundă/test

Focșani, 18-24 aprilie 2003

Olimpiada NAȚIONALĂ de INFORMATICĂ

Mihai Stroe

Vă prezentăm în continuare soluțiile problemelor propuse la ONI 2003, la clasele a IX-a și a X-a. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de autorii problemelor și au fost verificate folosind seturile de date cu care au fost testate soluțiile concurenților.

Clasa a IX-a

P050313: *Seti*

Rezolvarea problemei constă în realizarea unei metode de trecere de la cuvintele de 4 litere la numerele de cel mult 4 cifre. Având o astfel de metodă, toate cuvintele se vor transforma în numere cuprinse între 0 și 9999. Numerele nu vor fi memorate, în schimb se va folosi un tablou cu 10.000 de elemente, fiecare element, reprezentând numărul de apariții al unui cuvânt din fișier. În final, pentru fiecare număr de la 0 la 9999 se va afișa cuvântul reprezentat, de atâtea ori de câte indică numărul său de apariții.

Pentru a trece de la cuvânt la număr și invers se poate folosi ca etapă intermediară un tablou de patru cifre, transformările realizându-se foarte ușor.

Analiza complexității

Fie N numărul de cuvinte din fișier care poate avea valoarea maximă 200.000.

Operația de citirea a datelor are ordinul de complexitate $O(N)$. Metodele care transformă cuvintele în numere și invers au ordinul de complexitate $O(1)$. Actualizarea tabloului care contorizează numărul de apariții se face concomitent cu citirea, deci această operație are ordinul de complexitate $O(N)$. Operația de afișarea a datelor are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare este $O(N) + O(N) + O(N) = O(N)$.

P050314: *Circular*

O rezolvare foarte bună a acestei probleme se bazează pe faptul că există foarte puține numere circulare între 10 și 10.000.000. Acestea pot fi generate în timpul concursului, obținându-se un fișier cu toate numerele circulare, câte 10 pe linie și separate prin virgulă. Acest fișier poate fi transformat ușor într-un vector de constante care poate fi folosit de un program *Pascal* sau *C/C++* care, astfel conține toate numerele circulare. Acest program va citi valoarea lui N, apoi va rezolva cerințele parcurgând vectorul de constante. Deci, concurentul va elabora două programe: cel de generare a numerelor circulare și cel care, folosindu-le, rezolvă cerințele problemei.

Pentru generarea numerelor circulare se poate folosi un ciclu **for** până la 10.000.000, în cadrul căruia se testează, pe rând, pentru fiecare număr dacă este circular. Un mod simplu de a realiza această testare este de a transforma numărul într-un vector de cifre, prin împărțiri succesive la 10 cu salvarea restului.

Se observă că se vor obține cifrele numărului în ordine inversă; este foarte ușor să se restabilească ordinea corectă prin inversarea primei cifre cu ultima, celei de-a doua cu penultima etc.

Pentru un număr reprezentat ca un vector de cifre se poate verifica foarte ușor dacă este format din cifre distincte și nu conține cifra 0; pentru condiția rămasă se va număra spre dreapta, iar pozițiile pe care se oprește numărarea vor fi marcate cu 0. Dacă la un moment dat se va ajunge pe o poziție marcată cu 0, fără a se fi marcat în prealabil toate pozițiile, atunci numărul nu este circular.

O metodă mai bună și mai rapidă ar consta în generarea tuturor vectorilor de cifre distincte nenule de cel mult 7 cifre, folosind metoda *backtracking*.

O altă metodă, și mai rapidă, ar consta în generarea directă a numerelor circulare, folosind o altă variantă a metodei *backtracking*.

Astfel, după încercarea de a folosi cifra c_1 pe prima poziție, se va trece la poziția $1 + c_1$ (de fapt se vor face c_1 avansări spre dreapta) și se va continua de acolo; dacă acolo se va folosi cifra c_2, se avansează încă c_2 poziții spre dreapta etc.

La fiecare pas din *backtracking* se folosesc cifrele cuprinse între 1 și 9, care nu au fost folosite anterior; dacă avansarea spre dreapta duce într-o poziție deja completată și mai sunt poziții libere, numărul în construcție nu poate fi circular și se revine la cifra precedentă.

Analiza complexității

Pentru această problemă, complexitatea rezolvării este dată de numărul M de numere circulare, deci soluția rulează instantaneu, deci ordinul de complexitate al acesteia ar fi $O(\log_2 M)$, în cazul în care se folosește o căutare binară. Deoarece M este o constantă, ordinul de complexitate al soluției este $O(1)$.

Complexitatea metodei de generare a numerelor circulare depinde de metoda folosită.

P050315: Scaune

Rezolvarea acestei probleme constă în simularea deplasării copiilor, conform regulilor din enunț.

Deoarece limita pentru numărul de scaune este destul de mică, avansarea copiilor se poate realiza pas cu pas; astfel, în momentul în care un scaun se eliberează, fiecare copil avansează câte o poziție până când unul dintre ei ajunge în dreptul scaunului și îl ocupă.

O metodă mai rapidă constă în calcularea, pentru fiecare copil, a numărului de mutări până la ajungerea la scaunul liber. Fie un copil situat în poziția C, iar scaunul liber în poziția S; atunci copilul va avansa $S - C$ poziții, dacă $C \le S$, respectiv $NS + S - C$ poziții, dacă $C > S$. Astfel, la fiecare ridicare a unui copil se poate afla cel mai apropiat copil, se calculează câte poziții se va deplasa acesta (să spunem P), apoi toți copiii vor avansa P poziții. Un copil care avansează P poziții din poziția C va ajunge în poziția $C + P$ (dacă $C + P \le NS$), respectiv $C + P - NS$ (dacă $C + P > NS$).

Analiza complexității

Deoarece sunt NC copii în așteptare, vor fi analizate cel mult NC ridicări de pe scaun, indiferent dacă mai urmează ceva în fișierul de intrare. Dacă sunt mai puțin de NC copii care se ridică, atunci o parte dintre cei care așteaptă vor rămâne în picioare.

La fiecare moment în care se ridică un copil, avansarea pas cu pas a copiilor aflați în așteptare până la așezarea pe scaun a unuia dintre ei nu poate necesita mai mult de NS pași; fiecare pas necesită avansarea tuturor copiilor, deci a cel mult NC copii. Complexitatea ocupării unui scaun prin avansare pas cu pas este deci $O(NS \cdot NC)$.

În cazul metodei mai rapide, calcularea distanței pentru toți copiii până la scaunul liber are complexitatea $O(NC)$. Aceeași complexitate au și operațiile de alegere a minimului P dintre aceste distanțe, respectiv de avansare a tuturor copiilor cu P poziții.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(NC \cdot NS \cdot NC)$ pentru prima variantă, respectiv $O(NC \cdot NC)$ pentru cea de-a doua.

P050316: Criptare

Pentru decriptarea mesajului se utilizează algoritmul de criptare descris în sens invers. Se determină ordinea literelor din cuvântul cheie. Considerând că acest cuvânt are lungimea L, iar textul are lungimea N, el va fi împărțit în L grupe de lungimea N/L. Grupele sunt așezate într-o matrice cu L coloane pe coloanele corespunzătoare ordinii literelor din cuvântul cheie. În final, matricea este parcursă pe linii și se afișează mesajul decriptat.

Analiza complexității

Operațiile de citirea și afișare a datelor au ordinul de complexitate $O(N)$. Deoarece lungimea L a cuvântului cheie este mică, operațiile cu acesta (determinarea ordinii literelor) pot fi neglijate la calculul complexității. Scrierea cuvântului în matrice are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N) = O(N)$.

P050317: Mașină

Având în vedere că orașele sunt așezate și parcurse circular, putem dubla orașele, iar apoi putem să le considerăm așezate liniar:

$$a_n + 1 = a_1, a_n + 2 = a_2, \ldots, a_n + k = a_k, \ldots, a_{2n} = a_n$$
$$b_n + 1 = b_1, b_n + 2 = b_2, \ldots, b_n + k = b_k, \ldots, b_{2n} = b_n.$$

Cantitatea de benzină câștigată în urma opririi în orașul i și parcurgerii drumului către orașul $i + 1$ este $c_i = a_i - b_i$. Evident, această cantitate poate fi negativă.

Definim șirul sumelor parțiale ale vectorului c astfel:

$$s_i = \sum_{k=1}^{i} c_i.$$

Pentru ca *PAM* să poată efectua turul, este necesar ca suma cantității de benzină acumulate să fie mai mare sau egală cu 0, deci $s_n \ge 0$.

Rezolvarea problemei constă în a găsi un x astfel încât $c_x \ge 0, c_x + c_{x-1} \ge 0, \ldots, c_x + \ldots c_{x+n-1} \ge 0$, adică $s_x - s_{x-1} \ge 0, s_{x+1} - s_{x-1} \ge 0, \ldots, s_{x+n-1} - s_{x-1} \ge 0$. Un candidat viabil este poziția $m \le n$ pentru care s_m este minim. În continuare se demonstrează că această alegere conduce la o soluție.

Fie $m < k \le n$; avem $s_k \ge s_m$ (din definiția lui m).

Fie $n < k$; avem $s_k = c_1 + \ldots + c_n + c_{n+1} + \ldots + c_k = s_n + c_1 + \ldots + c_k = s_n + s_k \ge s_k \ge s_m$ (am folosit faptul că $s_n \ge 0$). În concluzie $x = m + 1$ este o soluție pentru problemă.

Implementarea constă în citirea datelor, calcularea șirului sumelor parțiale și găsirea minimului dintre $s_1, s_2, \ldots, s_n$.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(n)$.

Calculul șirului sumelor parțiale are ordinul de complexitate $O(n)$ pentru că $s_i = s_{i-1} + a_i$. Minimul din acest șir se calculează cu aceeași complexitate. Operația de scriere a rezultatului are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n) + O(n) + O(1) = O(n)$.

P050318: Operații

Pentru rezolvarea problemei este necesară descompunerea în binar a celor două numere și înțelegerea efectului pe care îl au cele două operații asupra acestei descompuneri.

Se observă că operația O_1 inserează o cifră 1 în descompunere, pe o anumită poziție, în timp ce operația O_2 șterge un 0, la alegere. Privind astfel cele două operații, algoritmul de rezolvare începe cu descompunerea numerelor în baza 2 și ștergerea tuturor cifrelor 0 din ambele numere. Se vor obține astfel două numere, nu neapărat egale, ale căror descompuneri în baza 2 sunt formate numai din cifre 1 (deci numerele sunt de forma $2 \cdot P - 1$). Cum cifrele 1 nu pot fi șterse, singura variantă de a ajunge la același număr constă în inserarea de cifre 1 în cadrul numărului mai mic.

La efectuarea operațiilor O_2 pe fiecare număr, acestea se memorează pentru a fi afișate. Pentru operațiile O_1 nu este necesară memorarea; ele se vor efectua numai asupra numărului care rămâne mai mic, valoarea pentru K putând fi 1 pentru toate aceste operații.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(1)$. De asemenea, ordinul de complexitate al operațiilor de descompunere a numerelor în baza 2, de detectare a zerourilor din descompunere, de memorare a pozițiilor operațiilor de tip 2, de detectare a numărului mai mic și afișarea mutărilor este $O(1)$.

În final ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

Clasa a X-a

P050319: Asediu

Această problemă se rezolvă matematic. Considerând poligonul format din cele n puncte fără a trasa diagonalele, fiecare latură formează cu circumferința cercului n regiuni, plus o regiune interioară poligonului.

Trasând pe rând fiecare diagonală, se adaugă un număr de regiuni egal cu unu plus numărul de intersecții cu diagonalele duse deja. Astfel, fiecare intersecție adaugă o regiune la numărul total de regiuni.

Considerând fiecare grup de 4 puncte, se observă că acesta determină exact o intersecție. În total sunt C_n^4 grupuri de câte 4 puncte. Numărând și diagonalele, se ajunge la:

$$nr.\ total\ de\ regiuni = n + 1 + \frac{n \cdot (n-3)}{2} + C_n^4 = 1 + C_n^2 + C_n^4.$$

În implementare se ține cont de faptul că se ajunge la numere mari. Dacă se folosește formula finală, se poate scăpa de implementarea împărțirilor pe numere mari, simplificând fracțiile rezultate înainte de a efectua produsele.

De exemplu: $(18 \cdot 17 \cdot 16 \cdot 15) / (2 \cdot 3 \cdot 4 \cdot 5)$ devine $3 \cdot 17 \cdot 4 \cdot 15$ etc.

Analiza complexității

Deoarece formula finală este un polinom de gradul 4 în funcție de N, calcularea rezultatului necesită un număr constant de operații, deci ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

P050320: Muzeu

Se observă că numărul variantelor de cumpărare a biletelor (cumpăr / nu cumpăr biletul 1, biletul 2,... biletul 10) este $2^{10} = 1024$.

Se generează fiecare din aceste variante. Pentru o astfel de variantă, se calculează costul și se transformă matricea inițială într-o matrice cu 0 și 1, în care 0 reprezintă o cameră pentru care se plătește biletul (sau nu e nevoie de bilet) și 1 reprezintă o cameră pentru care nu se cumpără bilet (deci în care nu se intră).

Pentru o astfel de matrice, problema determinării celui mai scurt drum de la (1, 1) la (M, N) se rezolvă cu algoritmul lui *Lee*.

Se rezolvă această problemă pentru toate cele 1024 variante. Eventual, dacă la un moment dat există o soluție cu un cost mai bun decât al variantei curente, aceasta nu mai este abordată.

Evident, problema determinării celui mai scurt drum se poate rezolva și pe matricea inițială, ținând cont la fiecare pas de biletele cumpărate; cum algoritmul lui *Lee* este folosit de obicei pentru o matrice cu 0 și 1, am folosit inițial convenția respectivă pentru a ușura înțelegerea

Se alege soluția de cost minim și, în caz de egalitate, cea cu număr minim de camere.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(M \cdot N)$.

Fie C numărul de numere de marcaj diferite din matrice. Pentru fiecare dintre cele 2^C variante, găsirea drumului minim cu algoritmul lui *Lee* are complexitatea $O(M \cdot N)$.

Operația de scriere a soluției are ordinul de complexitate $O(M \cdot N)$ pentru cazul cel mai defavorabil.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(M \cdot N \cdot 2^C)$.

Algoritmul funcționează datorită restricției impuse pentru C (cel mult 10 numere de marcaj). Considerând 2^C o constantă, ordinul de complexitate devine $O(M \cdot N)$.

P050321: Munte

Problema se rezolvă prin metoda *programării dinamice*.

Practic, problema se poate împărți în două subprobleme. Primul pas constă în determinarea perechilor de vârfuri care pot fi unite printr-un cablu. Acest pas se rezolvă folosind cunoștințe elementare de geometrie plană (ecuația dreptei, $y = a \cdot x + b$). Se va obține o matrice OK, unde $OK_{i,j}$ are valoarea 1 dacă vârfurile i și j pot fi unite și 0 în caz contrar. Folosind această matrice, se vor conecta vârfurile 1 și N cu un lanț de K stații, astfel încât oricare două stații consecutive să aibă OK-ul corespunzător egal cu 1.

Această subproblemă se rezolvă prin metoda *programării dinamice*, după cum urmează: se construiește o matrice A cu K linii și N coloane, unde $A_{i,j}$ reprezintă lungimea totală minimă a unui lanț cu i stații care conectează vârfurile 1 și j. Inițial $A_{1,1} = 0$; $A_{1,i} = +\infty$ și $A_{i,1} = +\infty$ pentru $i > 1$. Pentru i cuprins între 2 și N, componentele matricei se calculează astfel:

$Ai,j = \min(Ai\text{-}1,v + distv,j)$, unde $v < j$ și $OKv,j = 1$.

Concomitent cu calculul elementelor $A_{i,j}$ se construiește o matrice T, unde $T_{i,j}$ reprezintă valoarea lui v care minimizează expresia de mai sus. Matricea T este folosită pentru reconstituirea soluției.

Lungimea totală minimă este regăsită în $A_{K,N}$.

Subproblemele puteau fi tratate și simultan, ceea ce complica implementarea.

Analiza complexității

Operația de citire a datelor are ordinul de complexitate $O(N)$. Calculul matricei OK are ordinul de complexitate $O(N^3)$. Algoritmul bazat pe metoda *programării dinamice* are ordinul de complexitate $O(N^2 \cdot K)$. Reconstituirea soluției și afișarea au ordinul de complexitate $O(N)$.

Deoarece $K \leq N$, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N^3)$.

P050322: Partiție

Problema se poate rezolva în mai multe moduri.

O idee bună de rezolvare a problemei constă în folosirea metodei *programării dinamice*. Se poate construi o matrice A, unde $A_{i,k}$ reprezintă numărul de partiții ale numărului i cu numere impare, din care ultimul este cel mult egal cu k.

Un element din A se calculează observând că o partiție a lui i cu numere impare, cel mult egale cu k, este formată dintr-un un număr M, mai mic sau egal cu k, și alte numere, mai mici sau egale cu M. De aici rezultă relația:

$$A_{i,k} = \sum_{\substack{M=1,k;M\leq i;\\ M \text{ impar}}} A_{i-M,M}.$$

Inițial $A_{0,0}$ este 1. La implementare, pentru a economisi spațiu, se poate alege o variantă în care $A_{i,k}$ să reprezinte numărul de partiții al lui i cu numere impare, din care ultimul este cel mult egal cu al k-lea număr impar, adică $2 \cdot k - 1$.

După calcularea elementelor matricei, soluția pentru numărul i se găsește în $A_{i,i}$. Aceste valori pot fi salvate într-un vector de constante, care este transformat într-un nou program, în același mod ca la problema *Circular* de la clasa a *IX*-a. Această metodă conduce la o rezolvare instantanee a testelor date în concurs.

O altă metodă, bazată pe vectorul de constante, ar fi însemnat generarea soluțiilor folosind metoda *backtracking*. Un *backtracking* lăsat să se execute în timpul concursului ar fi putut genera soluțiile pentru toate valorile lui N, după care se putea folosi metoda vectorului de constante, în timp ce un *backtracking* folosit ca soluție a problemei ar fi obținut punctajul maxim doar pentru testele mici și medii (pentru valori ale lui N mai mici decât 130). Limitele problemei și timpul de execuție de 3 secunde permit rezolvării prin backtracking obținerea unui punctaj mulțumitor.

Analiza complexității

Pentru o rezolvare care se bazează pe metoda vectorului de constante, ordinul de complexitate al soluției finale ar fi fost $O(1)$; soluția constă în citirea valorii lui N și afișarea rezultatului memorat.

Soluția descrisă anterior, bazată pe metoda *programării dinamice*, are ordinul de complexitatea $O(n^3)$. Invităm cititorul să caute metode mai eficiente.

Ordinul de complexitate al unei soluții care se bazează pe metoda *backtracking* este exponențial.

P050323: Rubine

Prima parte a problemei constă în determinarea, pentru fiecare hoț, a numărului maxim de rubine pe care le poate strânge, respectând cerințele enunțului. Acest număr se poate determina, pentru fiecare hoț, plecând din poziția de start a acestuia și folosind un algoritm de complexitate $O(L \cdot C)$.

Putem optimiza această metodă dacă se pornește din poziția destinație (comună) a tuturor hoților; astfel acest algoritm se va executa o singură dată, obținându-se după executare rezultatul cerut pentru fiecare poziție de start posibilă.

În prima fază a algoritmului (amintit mai sus) se obține pentru fiecare poziție (i, j) numărul minim de mutări $M(i, j)$ până la ajungerea la destinație, pornind din poziția respectivă. Pentru aceasta putem folosi algoritmul lui *Lee*, de complexitate $O(L \cdot C)$.

O examinare mai atentă a numerelor rezultate conduce la un algoritm mult mai simplu, de aceeași complexitate. Astfel, $M[L, C] = 0$, $M[L - 1, C] = M[L - 1, C - 1] = M[L, C - 1] = 1$, ..., $M[L - d_1, C - d_2] = \max(d_1, d_2)$.

Folosind această informație vom calcula, pentru fiecare poziție (i, j) din matrice, numărul maxim de rubine $R(i, j)$ care poate fi strâns pornind spre destinație din poziția respectivă. Astfel, pentru o poziție (i, j), $R(i, j)$ este maximul dintre R-urile vecinilor care au M-ul mai mic cu 1 decât $M(i, j)$, la care se adaugă cantitatea de rubine din poziția (i, j).

Pozițiile pot fi parcurse în ordinea numerelor $M(i, j)$; astfel, pentru fiecare poziție, R-urile vecinilor care ne interesează vor fi calculate anterior, deci vor fi disponibile la calculul $R(i, j)$.

Deoarece pentru fiecare poziție din matrice se vor examina cel mult trei vecini, calcularea matricei R are complexitatea $O(L \cdot C)$.

Se folosește apoi un algoritm *greedy* pentru selectarea perechii (i, j), j fiind un hoț pentru care nu există perechi (j, k).

Ordinul de complexitate al unei implementări pentru acest algoritm este $O(N \cdot N)$; o implementare mai preten-

ţioasă poate avea ordinul de complexitate $O(P)$, unde P este numărul de perechi (i, j) prezente în fişierul de intrare.

Limitele din enunţ permit şi variante mai puţin optime, cum ar fi cea în care algoritmul lui *Lee* este executat pentru fiecare hoţ.

Analiza complexităţii

Ordinul de complexitate al operaţiei de citire a datelor de intrare este $O(L \cdot C + N + P)$. Deoarece P este limitat superior de $N \cdot N$, ordinul de complexitate al acestei operaţii devine $O(L \cdot C + N \cdot N)$.

Calcularea cantităţii de rubine strânse de fiecare hoţ are ordinul de complexitate $O(L \cdot C)$.

Alegerea perechilor pentru repartizarea sacilor are ordinul de complexitate $O(P)$ sau $O(N \cdot N)$, în funcţie de implementarea aleasă.

Afişarea soluţiei are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(L \cdot C + N \cdot N) + O(N \cdot N) + O(N) = O(L \cdot C + N \cdot N)$.

P050324: *Scufiţa*

Problema se rezolvă folosind metoda *backtracking* în plan. Se observă că mutările *Lupului* sunt independente de mutările *Scufiţei*, astfel ele pot fi determinate imediat după citirea datelor.

În acest punct al rezolvării, *Scufiţa* trebuie să mute la fiecare moment, astfel încât să rămână în joc, iar în final să strângă cât mai multe ciupercuţe. Restricţiile sunt date de enunţul problemei şi de numărul de ciupercuţe culese de *Lup*, care este cunoscut la fiecare moment.

Rezolvarea prin metoda *backtracking* încearcă la fiecare pas, pe rând, fiecare mutare din cele cel mult patru posibile (teoretic; de fapt, cel mult trei mutări sunt posibile în fiecare moment, deoarece *Scufiţa* nu se va întoarce în poziţia din care tocmai a venit).

Se urmăreşte respectarea restricţiilor impuse. În momentul găsirii unei soluţii, aceasta este comparată cu soluţia optimă găsită până atunci şi dacă este mai bună este reţinută.

Încă nu a fost găsită o rezolvare polinomială pentru această problemă (şi este improbabil ca o astfel de rezolvare să existe), dar limitele mici şi faptul că numărul de mutări disponibile începe să scadă destul de repede, în multe cazuri permit un timp de execuţie foarte bun.

Analiza complexităţii

Fie M numărul de mutări pe care le are de efectuat *Scufiţa*.

Operaţiile de citire şi scriere a datelor au ordinul de complexitate $O(N^2)$, respectiv $O(M)$, deci nu vor fi luate în calcul. Ordinul de complexitate este dat de rezolvarea prin metoda *backtracking*.

Având în vedere că la fiecare pas *Scufiţa* poate alege dintre cel mult 3 mutări (deoarece nu se poate întoarce în poziţia din care a venit), ordinul de complexitate ar fi $O(3^M)$. De fapt, numărul maxim de stări examinate este mult mai mic; de exemplu, primele două mutări oferă două variante de alegere în loc de trei. Alte restricţii apar datorită limitării la o matrice de $10 \cdot 10$ elemente.

Cum numărul M este cunoscut şi mic, s-ar putea considera că ordinul de complexitate este limitat superior, deci constant (constanta introdusă fiind totuşi destul de mare).

AMD64

Pe data de 23 septembrie, compania *AMD* a lansat pe piaţă mult aşteptatele modele de procesoare pe 64 de biţi destinate calculatoarelor de birou. Noile procesoare, *Athlon 64 3200+* şi *Athlon 64 FX-51* se alătură celorlalte procesoare care folosesc tehnologia AMD64, cele din familia Opteron, care sunt destinate serverelor.

Indicativul 3200+ al procesorului *Athlon 64* arată că performanţele sale sunt comparabile cu cele ale unui procesor realizat de compania *Intel* care are o frecvenţă de 3,2 *GHZ*. *Athlon 64* are o frecvenţă de 2 *GHZ*, foloseşte o magistrală cu frecvenţa de 400 *MHZ*, iar interfaţa sa externă este *Socket 754*.

Procesorul *Athlon 64 FX-51*, spre deosebire de *Athlon 64 3200+*, are frecvenţa de 2,2 *GHZ*, iar performanţele sunt asemănătoare cu cele ale unui procesor Intel Pentium 4 Extreme Edition cu frecvenţa de 3,2 GHz (acesta este un procesor Xeon modificat pentru a fi folosit pentru calculatoarele de birou, performanţele sale superioare fiind datorate unei memorii cache de nivel 3, cu dimensiunea de 2 MB).

Noile modele de procesoare lansate de *AMD* sunt echipate cu diode termosensibile care înregistrează schimbările de temperatură de 16 ori pe secundă şi întrerup funcţionarea procesorului dacă temperatura depăşeşte valorile normale.

Setul de instrucţiuni al celor două modele de procesoare este format din instrucţiunile procesoarelor din familia *Athlon XP* pe 32 de biţi la care s-au adăugat instrucţiuni pe 64 de biţi, precum şi instrucţiunile SSE2.

O altă facilitate de care dispun aceste procesoare este tehnologia *HyperTransport*. Cu ajutorul acesteia, sistemele dotate cu aceste procesoare sunt mult mai eficiente, fiind diminuate pierderile de performanţă datorate transferurilor de date între acestea şi placa de bază.

Etapa FINALĂ

Ediția a patra a concursului de programare organizat de Gazeta de Informatică s-a încheiat. Primii clasați după cele 20 de runde desfășurate prin e-mail au participat la faza finală care a avut loc pe data de 17 mai 2003 la Liceul "Emil Racoviță" din Cluj-Napoca.

Dintre cei zece concurenți care s-au calificat la faza finală, doar șapte au reușit să se prezinte la concurs.Aceștia sunt:

Ștefan Ciobâcă, Suceava
Silvestru-Cosmin Negruşeri, Bistrița
Ciprian Cană, Vaslui
Adrian Cârcu, Bistrița
Tiberiu Chiorean, Baia Mare
Csaba Pătcaș, Oradea
Aurelian Marinescu, Brașov

Bursa de 50$/lună oferită de editura *Agora Media* a fost câștigată de *Ștefan Ciobâcă*. Pe locul al doilea s-a clasat concurentul de origine rusă *Ivan Kazmenko* care, datorită unor dificultăți nu a putut ajunge în România, dar a participat *on-line* la etapa finală. Deoarece nu s-au acordat burse concurenților străini, bursa de 30$/lună a fost câștigată de *Silvestru-Cosmin Negruşeri*.

Toți participanții la finală au primit cărți oferite de *Agora Media* și *Computer Libris Agora*.

În continuare vă prezentăm enunțurile celor patru probleme propuse spre rezolvare la etapa finală a celei de-a patra ediții a concursului de programare organizat de revista noastră.

P060314: Dragoni

Noul maestru al dragonilor controlează un număr de N puncte de observație ale căror coordonate se cunosc. Nu există trei puncte de observație ale căror coordonate sunt puncte coliniare. Pentru a se proteja de următorul atac al megadragonilor, el decide să trimită câte un dragon la fiecare dintre punctele de observație aflate pe perimetrul teritoriului. Acest perimetru este dat de poligonul de perimetru minim care conține în interior sau în vârfuri toate punctele de observație.

Datorită faptului că maestrul dragonilor știe că, mai devreme sau mai târziu, acest perimetru va fi străpuns, el organizează apărarea în continuare astfel:

- punctele de observație aflate pe perimetru (în vârfurile poligonului) sunt considerate a fi pierdute;
- pentru punctele de observație rămase se determină un nou perimetru și la punctele de observație aflate pe noul perimetru sunt trimiși câte doi dragoni;
- evident, și acest perimetru va fi străpuns, deci este organizat un al treilea nivel de apărare care va fi dat de perimetrul punctelor de observație rămase; la punctele de observație de pe acest perimetru vor fi trimiși câte trei dragoni;
- procedeul va continua (numărul dragonilor crește cu unu pentru fiecare nou perimetru) până în momentul în care nu va mai rămâne nici un punct de observație.

Va trebui să determinați, pentru fiecare punct de observație în parte, numărul dragonilor care vor fi trimiși la punctul de observație respectiv.

Date de intrare

Prima linie a fișierului de intrare **DRAGONS.IN** conține numărul N al punctelor de observație de pe teritoriul maestrului dragonilor. Fiecare dintre următoarele N linii va conține o pereche de numere reale, reprezentând coordonatele unui punct de observație, separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **DRAGONS.OUT** trebuie să conțină N linii; fiecare linie corespunde unui punct de observație. O astfel de linie va conține un singur număr care reprezintă numărul dragonilor trimiși la punctul de observație respectiv. Ordinea punctelor de observație din fișierul de ieșire trebuie să respecte ordinea acestora din fișierul de intrare.

Restricții

- $3 \le n \le 1000$;
- coordonatele punctelor de observație sunt numere reale cu cel mult trei zecimale exacte;
- eventual, ultimul "perimetru" poate conține unul sau două puncte de observație.

Exemplu

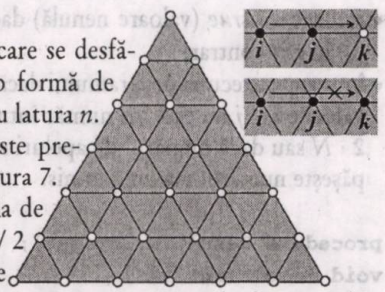
DRAGONS.IN	DRAGONS.OUT
16	1

8 2	1
2 4	1
17 4	2
6 5	3
8 7	3
11 9	2
15 10	2
12 18	2
5.1 16	1
4 20	1
15 18	3
7 15	3
6 14	4
11 13	4
12 13	3
13 14	

Timp maxim de execuție/test: 1 secundă

P060315: Joc

Se consideră un joc care se desfășoară pe o tablă sub formă de triunghi echilateral cu latura n. În figura alăturată este prezentată o tablă cu latura 7. Se observă că pe tabla de joc se află $n \cdot (n + 1) / 2$ puncte de intersecție în care se află scobituri. La început, în unele dintre scobiturile de pe tabla de joc se vor afla bile. Scobiturile vor fi numerotate de la 1 la $n \cdot (n + 1) / 2$ de sus în jos și de la stânga la dreapta. Jocul constă din mai multe mutări și pentru fiecare mutare există o singură regulă: o bilă de pe poziția i poate sări peste o bilă de pe poziția j și se va pune într-o locație liberă k vecină poziției j, dacă poziția i este vecină cu poziția j; pozițiile i, j și k se află pe același segment. După efectuarea unei astfel de mutări bila de pe poziția j va fi eliminată de pe tabla de joc. Procedeul continuă până în momentul în care nu se mai pot efectua mutări.



Cunoscând latura n a tablei de joc și amplasamentul inițial al bilelor, se cere să se determine mutările care trebuie efectuate, astfel încât să se ajungă într-o configurație care conține cât mai puține bile.

Date de intrare

Fișierul de intrare **GAME.IN** conține pe prima linie două numere întregi strict pozitive n și m separate printr-un singur spațiu, care reprezintă latura tablei de joc, respectiv numărul de scobituri de pe tabla de joc în care se află bile. Pe următoarea linie se află m numere întregi strict pozitive separate prin câte un spațiu, care reprezintă numerele de ordine ale scobiturilor care conțin bile.

Date de ieșire

Fișierul de ieșire **GAME.OUT** va conține cel mult m - 1 linii. Pentru fiecare mutare se va se afla pe câte o linie, o pereche formată din două numere naturale strict pozitive i și j, separate prin spațiu, care reprezintă numărul de ordine al scobiturii din care se ia o bilă și numărul de ordine al scobiturii care conține bila peste care se sare. Mutările se vor scrie în fișier în ordinea în care se efectuează.

Restricții

- $4 \le n \le 20$;
- $1 \le m < n \cdot (n + 1) / 2$.

Exemplu

GAME.IN	GAME.OUT
4 5	8 5
4 5 6 7 8	7 4
	6 3
	1 2

Modalitatea de acordare a punctajului

Vom considera că pentru fiecare test se vor putea obține cel mult X puncte. Concurenții care vor obține cea mai mică valoare $NrMin$ pentru numărul de bile rămase vor primi X puncte pentru testul respectiv.

Ceilalți concurenți, care au efectuat mutări corecte, dar numărul total al bilelor rămase pe tablă este $Nr > NrMin$, vor obține $X \cdot NrMin / Nr$ puncte pentru testul respectiv. Această valoare va fi aproximată cu două zecimale exacte.

Punctajul final va fi obținut prin adunarea punctajelor de la fiecare test și rotunjirea acestuia la cel mai apropiat număr întreg. Dacă mutările efectuate nu sunt corecte, concurenții nu vor primi nici un punct pentru testul respectiv.

De exemplu, dacă pentru un test se pot obține cel mult 5 puncte, cel mai bun rezultat obținut de un concurent constă într-o succesiune de mutări în urma cărora pe tablă rămân 2 bile, iar un alt concurent obține o succesiune de mutări în urma cărora pe tablă rămân 4 bile, atunci, pentru testul respectiv, punctajul concurentului va fi de $5 \cdot 2 / 4 = 2{,}5$ puncte.

Timp maxim de execuție/test: 1 secundă

P060316: Produse

Se consideră o matrice pătratică A de dimensiune N ale cărei elemente sunt date de formula $a_{i,j} = i^2 \cdot j$. Un produs va fi obținut prin alegerea unei permutări p a mulțimii $\{1, 2, ..., N\}$ și înmulțirea tuturor elementelor de forma $a_{i,x}$, unde $x = p_i$.

Va trebui să determinați numărul de cifre, în baza 10, al sumei tuturor produselor distincte posibile. Două produse se consideră a fi diferite dacă permutările corespunzătoare sunt diferite.

Date de intrare

Fișierele de intrare sunt deschise, iar numele acestora au forma **PRODUCTSXX.IN**, unde **XX** ia valori între 01 și 20 și reprezintă numărul testului.

Fiecare dintre cele 20 de fișiere de intrare conține o singură linie pe care se află valoarea N.

Date de ieșire

Pentru această problemă nu va trebui să trimiteți un program care să o rezolve, ci doar cele 20 de fișiere de ieșire corecte. Acestea vor fi denumite **PRODUCTSXX.OUT**, unde **XX** ia valori între 01 și 20 și reprezintă numărul testului.

Prima linie a fișierului de ieșire va conține textul **PRODUCTS - TEST #XX**, unde **XX** reprezintă numărul testului. Cea de-a doua linie va conține numărul cifrelor sumei tuturor produselor posibile.

Exemplu

Vom considera că acest exemplu reprezintă testul 00.

PRODUCTS00.IN	PRODUCTS00.OUT
2	2

Modalitatea de acordare a punctajului

Pentru fiecare fișier de ieșire corect veți obține un anumit număr de puncte. Punctajul maxim care poate fi obținut (dacă toate cele 20 de fișiere de ieșire sunt corecte) este de 100 de puncte. Pentru primele patru teste (01 - 04) se vor acorda câte 3 puncte, pentru următoarele patru (05 - 08) câte 4 puncte, pentru următoarele patru (09 - 12) câte 5 puncte, pentru următoarele patru (13 - 16) câte 6 puncte, iar pentru ultimele patru (17 - 20) câte 7 puncte.

Timp maxim de execuție/test: 1 secundă

P060317: Mijloc

Se consideră un șir a cu $2 \cdot N$ elemente ale căror valori sunt numere întregi distincte, generate aleator, care nu sunt cunoscute. Se știe că atât primele N elemente ale șirului (cele de pe pozițiile 1, 2, ..., N), cât și ultimele N elemente ale acestuia (cele de pe pozițiile $N + 1$, $N + 2$, ..., $2 \cdot N$) sunt ordonate crescător.

Va trebui să determinați unul dintre indicii elementelor aflate pe pozițiile N sau $N + 1$ în șirul obținut prin sortarea în ordine crescătoare a șirului a. Pentru determinarea acestui indice, puteți efectua un număr limitat de comparări între două elemente ale șirului a și aveți dreptul la două încercări de a "ghici" acest indice.

Pentru aceasta aveți la dispoziție o bibliotecă externă care vă pune la dispoziție rutine pentru:

- inițializare;
- determinarea numărului de elemente ale șirului;
- determinarea numărului maxim de comparări pe care le aveți la dispoziție;
- compararea a două elemente ale șirului a;
- furnizarea rezultatului.

Bibliotecile Pascal și C/C++

Pentru a putea folosi biblioteca va trebui să includeți în programul dumneavoastră instrucțiunea **uses** `middle;` pentru limbajul *Pascal* și `#include "middle.h"` pentru limbajul *C/C++*.

În continuare sunt prezentate sintaxele funcțiilor și procedurilor incluse în biblioteci:

```
procedure Init;
void Init()
```

- trebuie apelată la începutul programului;
- este folosită de către bibliotecă pentru inițializare și întrerupe execuția programului dacă este apelată a doua oară.

```
function Get2N:Integer;
int Get2N()
```

- returnează numărul de elemente ($2 \cdot N$) ale șirului a.

```
function GetNoComp:Integer;
int GetNoComp()
```

- returnează numărul de comparări (apeluri ale funcției `IsGreater`) pe care le mai aveți la dispoziție (în momentul apelului funcției `GetNoComp`) pentru determinarea rezultatului.

```
function IsGreater(i,j:Integer):Boolean;
int IsGreater(int i,int j)
```

- returnează *true* (valoare nenulă) dacă $a[i] > a[j]$ și *false* (0) în caz contrar;
- întrerupe execuția programului dacă cel puțin una dintre valorile i și j nu este un număr întreg cuprins între 1 și $2 \cdot N$ sau dacă numărul de apeluri ale acestei funcții depășește numărul maxim permis.

```
procedure Result(x:Integer);
void Result(int x)
```

- acceptă ca rezultat faptul că elementul de pe poziția x a șirului a se află pe una dintre pozițiile N sau $N + 1$ în șirul obținut după ordonare;
- întrerupe execuția programului dacă valoarea x reprezintă un rezultat corect sau dacă este apelată a doua oară și valoarea x nu reprezintă un rezultat corect.

Toate funcțiile și procedurile (excepție: `Init`) întrerup execuția programului dacă sunt apelate înaintea inițializării.

Programul vostru nu va citi date din nici un fișier de intrare și nu va scrie date în nici un fișier de ieșire.

Restricții și precizări

- $2 \le N \le 2457$;
- atât indicele elementului de pe poziția N, cât și indicele celui de pe poziția $N + 1$ sunt considerate rezultate corecte;
- doar unul dintre apelurile permise ale rutinei `Result` trebuie să furnizeze un răspuns corect.

Exemplu de utilizare a bibliotecii

Subprogram apelat	*Valoare returnată*
`Init`	-
`Get2N`	6
`GetNoComp`	2
`IsGreater(2,5)`	`false (0)`
`IsGreater(1,4)`	`false (0)`
`Result(5)`	-
`Result(4)`	-

Timp maxim de execuție/test: 1 secundă

REGULAMENT de participare

Ediția a V-a a concursului de programare organizat de Gazeta de Informatică începe pe data de 1 noiembrie 2003. Pentru detalii referitoare la acest concurs citiți paginile următoare sau vizitați secțiunea de pe site-ul nostru dedicată concursului: www.ginfo.ro/concurs.

Informații generale

Concursul *Bursele Agora* este o competiție internațională destinată elevilor și studenților. Acest concurs este organizat de redacția *GInfo* și editura *Agora Media*.

Prima fază a concursului constă în mai multe runde; la fiecare dintre ele participanții trebuie să rezolve câte o problemă și să trimită soluțiile prin *e-mail*; adresa de *e-mail* a concursului nostru este **concurs@ginfo.ro**.

După etapele care se vor desfășura prin *e-mail*, va fi o etapă finală care va avea loc la *Cluj-Napoca*; doar primii zece clasați în urma primei faze a concursului vor participa la faza finală. Cheltuielile de cazare și transport pe teritoriul României vor fi suportate de organizatori.

Problemele care trebuie rezolvate sunt propuse de redacția *GInfo* în colaborare cu studenți și profesori care se ocupă de pregătirea elevilor români care participă la diferite concursuri naționale și internaționale.

Premii

Concurentul clasat pe prima poziție în urma desfășurării fazei finale va fi premiat cu o bursă în valoare de 50$ pe lună, pe durata anului școlar/universitar următor.

Concurentul clasat pe a doua poziție va fi premiat cu o bursă în valoare de 30$ pe lună, pe durata anului școlar/universitar următor.

Așadar, cuantumul total al premiilor este de 80$ pe lună. La aceste premii se vor adăuga altele oferite de diferiți sponsori.

În cazul în care doi sau mai mulți concurenți se vor afla la egalitate pe primul loc, suma de 80$ va fi împărțită în mod egal între aceștia.

În cazul în care un singur concurent se clasează pe primul loc, dar doi sau mai mulți concurenți se află la egalitate pe locul al doilea, atunci suma de 30$ va fi împărțită în mod egal între aceștia din urmă, primul clasat fiind premiat cu 50$.

În cazul în care un concurent se clasează pe o poziție care îi dă dreptul la o bursă dar, potrivit secțiunii *Dreptul de participare* a prezentului regulament, el nu are dreptul să o câștige, atunci va fi eliminat din clasament.

Înscrierea

Pentru a participa la această ediție a concursului de programare *Bursele Agora* va trebui, mai întâi, să vă înscrieți. Cea mai simplă modalitate de înscriere este completarea *online* a formularului de înscriere (**www.ginfo.ro/concurs/inscriere.shtml**) care va fi trimis automat, prin intermediul clientului dumneavoastră de *e-mail*, la **concurs@ginfo.ro**.

O altă posibilitate o constituie trimiterea unui *e-mail* la adresa **concurs@ginfo.ro**. Acesta va trebui să conțină informațiile cerute în formularul disponibil *online* și anume: numele și prenumele dumneavoastră, data nașterii, adresa *e-mail*, adresa poștală și numărul de telefon.

Cea de-a treia posibilitate constă în trimiterea unei scrisori care va conține aceleași informații. Aceasta trebuie trimisă pe adresa:

Redacția GInfo
str. Universității nr. 8/8,
400091 Cluj-Napoca

În termen de cinci zile lucrătoare de la primirea de către redacția *GInfo* a *e-mail*-ului sau a scrisorii dumneavoastră, vă vom trimite un *e-mail* care va conține codul dumneavoastră de identificare pentru ediția 2003/2004 a concursului.

Dreptul de participare

La prima fază a concursului *nu au drept de participare* concurenții care au fost descalificați la una dintre precedentele patru ediții ale concursului.

De asemenea, nu au drept de participare autorii articolelor publicate în *GInfo* pe parcursul desfășurării concursului. În cazul în care este publicat un articol care are ca

autor sau coautor un anumit concurent, atunci concurentul nu are dreptul să participe la concurs, începând cu ziua primirii materialului la redacție.

Nu au drept de participare la concurs membrii redacției *GInfo*, angajații editurii *Agora Media*, precum și rudele de gradul I și II ale acestora.

Nu au drept de participare la *faza finală* a concursului de programare *Bursele Agora* care au participat deja la cinci finale ale acestuia.

Nu au dreptul de *a câștiga* una dintre *Bursele Agora* concurenții care au beneficiat deja de trei burse în anii anteriori.

Desfășurarea concursului

Limbajele de programare acceptate la concursul *Bursele Agora* sunt *Pascal* și *C/C++*. Fișierele sursă vor fi compilate folosind *Borland Pascal* 7.0 sau *Borland C++* 3.1.

Problemele vor fi publicate pe *site*-ul *Web* al revistei noastre într-o secțiune dedicată acestui scop (**www.ginfo.ro/concurs/probleme.shtml**). Rezolvările vor trebui trimise înaintea expirării termenului limită specificat pentru fiecare rundă.

O singură soluție va fi acceptată pentru fiecare problemă. Dacă un concurent va trimite mai multe soluții pentru o anumită problemă, numai prima va fi luată în considerare, restul fiind ignorate. Redacția *GInfo* își rezervă dreptul de a descalifica concurenții care trimit soluții identice (sau aproape identice).

Este interzisă folosirea de secvențe scrise în limbaj de asamblare; dacă un fișier sursă conține astfel de secvențe, concurentul va primi 0 puncte pentru problema de la runda respectivă.

Este interzis ca programele generate folosind sursele concurenților să modifice *timer*-ul. Concurenții care vor încerca să modifice ceasul sistem vor fi descalificați.

Dacă aveți neclarități referitoare la enunțurile problemelor, puteți adresa întrebări folosind adresa *e-mail* a concursului. Nu puteți adresa decât întrebări ale căror răspunsuri posibile sunt `DA` sau `NU`. Dacă răspunsul poate fi găsit în enunțul problemei sau întrebarea nu respectă condiția anterioară, atunci răspunsul la întrebare va fi `FĂRĂ COMENTARII`.

În cazul în care redacția *GInfo* consideră că răspunsul la o anumită întrebare trebuie primit de toți participanții, atunci cel care a adresat întrebarea va primi răspunsul `ANUNȚ GENERAL`. Anunțurile generale vor fi disponibile *online* în secțiunea dedicată problemelor.

Fiecare etapă va fi anunțată cu cel puțin 24 de ore înaintea momentului în care aceasta va începe. Anunțul va conține data și ora începerii etapei, precum și durata acesteia. Durata unei etape poate varia între o oră și 72 de ore.

Enunțul problemei va fi disponibil începând cu data și ora specificate în anunț. În acest moment va fi anunțat și termenul limită (data și ora) pentru trimiterea soluțiilor.

Nu vor fi acceptate soluțiile care ajung la redacție după expirarea termenului limită.

Trimiterea soluțiilor

Pentru a trimite soluțiile de la o anumită rundă va trebui să trimiteți un *e-mail* având ca subiect codul dumneavoastră de identificare urmat de textul `Runda XX`, unde `XX` reprezintă numărul rundei.

Acest *e-mail* va avea atașată o arhivă `.ZIP`, `.ACE` sau `.RAR` numită **`YYYYYRXX.EXT`**, unde `YYYYY` este codul dumneavoastră de identificare, `XX` este numărul rundei, iar `EXT` este extensia (`RAR`, `ACE` sau `ZIP`).

Arhiva va conține două fișiere. Primul dintre ele va fi un fișier `.BAT` care va conține linia de comandă pe care doriți să o folosim pentru a compila fișierul sursă (al doilea fișier al arhivei), corespunzător rundei respective. Acest prim fișier al arhivei se va numi **`YYYYYRXX.BAT`**, unde `YYYYY` este codul dumneavoastră de identificare, iar `XX` este numărul rundei.

Cel de-al doilea fișier din arhivă va conține codul sursă corespunzător rezolvării pe care doriți să o trimiteți pentru problema de la runda respectivă. Denumirea acestuia va fi **`YYYYYRXX.PAS`**, **`YYYYYRXX.C`** sau **`YYYYYRXX.CPP`**. Din motive de securitate, arhivele autoexpandabile nu sunt acceptate.

Liniile de comandă din fișierul `.BAT` trebuie să aibă efect doar asupra fișierului sursă trimis de concurenți. Dacă fișierul conține comenzi care manipulează alte fișiere sau alte operații, atunci redacția *GInfo* își rezervă dreptul de a-l avertiza sau descalifica pe concurentul respectiv.

Arhiva nu va conține nici un alt fișier sau director, deci nu puteți trimite fișiere auxiliare împreună cu fișierele sursă.

Dacă programul dumneavoastră trebuie să creeze fișiere auxiliare, atunci acestea vor avea extensia `.TMP`, vor fi create în directorul curent și trebuie șterse înaintea terminării execuției programului. Dacă această regulă nu este respectată concurentul va primi 0 puncte pentru problema corespunzătoare, chiar dacă soluția este corectă.

Dacă redacția *GInfo* consideră că un concurent pune în pericol buna desfășurare a competiției (prin suprascrierea unor fișiere, crearea de fișiere foarte mari etc.), atunci acel concurent poate fi descalificat.

Toate fișierele trimise devin proprietatea editurii *Agora Media* și nu pot fi folosite în scopuri comerciale fără acordul scris al editurii.

Pentru dezarhivarea fișierelor trimise de concurenți se vor folosi următoarele programe:

- *WinZip* 8.1;
- *WinRar* 3.2;
- *WinAce* 2.5.

La unele runde nu va trebui să trimiteți soluții sub forma unor coduri sursă, ci doar fișierele de ieșire corecte pentru anumite fișiere de intrare pe care le veți avea la dispoziție. Pentru aceste runde modalitatea de trimitere a soluțiilor va fi descrisă în enunțul problemei.

În cazul în care, pentru o anumită rundă, un concurent nu respectă modalitatea de trimitere a soluțiilor, atunci

redacția *GInfo* își rezervă dreptul de a modifica anumite fișiere pentru a putea testa soluția. Redacția *GInfo* nu își asumă nici o răspundere cu privire la efectele modificărilor asupra punctajului concurentului.

În cazul în care redacția *GInfo* efectuează astfel de modificări, concurentul va primi un avertisment. Dacă, la una din runde, un concurent deja avertizat nu respectă modalitatea de trimitere, atunci el nu va primi nici un punct la runda respectivă.

Evaluarea

Pentru fiecare problemă puteți primi maximum 100 de puncte. Pentru corectarea fiecărei probleme vom folosi între 5 și 20 de seturi de date; dacă, pentru un anumit set de date, fișierul de ieșire este corect, atunci concurentul va primi un număr de puncte care se va adăuga la numărul de puncte corespunzător problemei respective. În cazul anumitor probleme este posibilă acordarea de punctaje parțiale.

Evaluarea va fi automată; în cazul în care concurenții trebuie să trimită programe de rezolvare a problemelor, în vederea efectuării evaluării se va folosi întotdeauna ultima versiune a programului de evaluare *GInfo Evaluator*.

Calculatorul folosit pentru evaluare are următoarea configurație: procesor *AMD Athlon XP* 1700+ (frecvența reală: 1,47 GHz), RAM: 256 MB, HDD: 40 GB.

Este garantat faptul că în timpul evaluării vor fi liberi cel puțin 400 KB de memorie convențională și cel puțin 100 MB de spațiu pe disc. Sistemul de operare folosit în timpul evaluării va fi *Microsoft Windows XP Professional*. Dacă un program va efectua o operație ilegală, atunci concurentul va primi 0 puncte pentru setul de date corespunzător.

Sfaturi

Programatorii în *Pascal* nu ar trebui să folosească niciodată *unit*-ul `CRT`, deoarece programele lor se vor opri returnând eroarea de execuție `Error 200: Division by zero`. Acest fapt este cauzat de un bine cunoscut *bug* al mediului de programare *Borland Pascal* 7.0; nici o problemă propusă spre rezolvare în cadrul concursului *Bursele Agora* nu necesită folosirea acestui unit.

Atunci când sunt folosite anumite tipuri de date (cum ar fi `Comp`) modul `8087` trebuie să fie activ. Opțiunile de mediu permit setarea acestui mod ca mod implicit, dar de obicei acest mod nu este activ. Programatorii în *Pascal* sunt rugați să includă în codul sursă directiva de compilare `{$N+}`, dacă doresc să folosească tipuri de date care necesită activarea acestui mod. În caz contrar, în mometul în care vom încerca să compilăm fișierul sursă, va fi generată eroarea de compilare `Error 116: Must be in 8087 mode to compile this`. Pentru a vă asigura că programul se va compila exact ca pe calculatorul folosit pentru a-l scrie, apăsați `CTRL+O,O`. Astfel, în sursa dumneavoastră, vor fi incluse toate directivele de compilare cu opțiunile care sunt setate în mediu. Puteți modifica aceste opțiuni astfel încât să fie cele dorite de dumneavoastră. Nu vă îngrijorați dacă nu cunoașteți semnificația unor anumite directive de compilare; acestea nu vor avea efecte negative asupra execuției programului - dacă programul dumneavoastră va rula corect pe calculatorul pe care l-ați folosit pentru a-l scrie, atunci va rula corect și pe calculatorul folosit pentru evaluare.

Toate programele trebuie să citească datele de intrare din fișierul de intrare specificat. Nu ar trebui să fie așteptate date de la intrarea standard (tastatură). Dacă programul dumneavoastră va aștepta o reacție de la tastatură, atunci execuția sa va fi oprită după expirarea timpului și nu veți obține nici un punct pentru acea soluție, chiar dacă ea este corectă. Deci, vă rugăm să nu includeți în codul sursă secvențe de tipul `Readln`, `getch()`, `getche()` etc.

Toate programele trebuie să scrie datele de ieșire în fișierul de ieșire specificat. Puteți scrie date și la ieșirea standard, dar aceasta nu va duce decât la încetinirea programului, deci nu este recomandabil să scrieți date pe ecran. Toate datele scrise la ieșirea standard sunt ignorate în timpul evaluării.

Programul de evaluare va verifica numai soluția scrisă în fișierul de ieșire specificat. Fișierul de intrare se află întotdeauna în directorul curent; nu încercați să citiți date din alte locații. Fișierul de ieșire va trebui generat tot în directorul curent; nu scrieți date de ieșire în alte locații. Trebuie să folosiți întotdeauna denumirile specificate în enunț pentru fișierul sursă, fișierul de intrare și fișierul de ieșire. Folosirea altor denumiri va duce la funcționarea incorectă a programului.

Exemplu

Denumirea fișierului de intrare: **`SUMA.IN`**
Denumirea fișierului de ieșire: **`SUMA.OUT`**
Descriere: Dându-se două numere, aflați suma lor.
Date de intrare: Fișierul de intrare conține, pe o singură linie, cele două numere separate printr-un singur spațiu.
Date de ieșire: Fișierul de ieșire ar trebui să conțină suma celor două numere.
Exemplu:

`SUMA.IN`	**`SUMA.OUT`**
2 4	6

Timp maxim de execuție: 1 secundă/test

După citirea enunțului veți începe să rezolvați problema. Veți observa că nu este clar dacă numerele sunt întregi sau reale.

Pentru a clarifica acest lucru va trebui să adresați o întrebare. Aceasta nu poate fi: *Numerele sunt întregi sau reale?* deoarece nu se poate răspunde la ea prin `DA` sau `NU`. O soluție ar fi să întrebați: *Numerele sunt întregi?* Veți primi răspunsul `NU` și veți ști că numerele sunt reale.

Acum apare o întrebare mult mai importantă: *Ce precizie trebuie folosită pentru afișarea rezultatului?* Se pare că redacția *GInfo* a uitat să precizeze acest lucru în enunț. Bineînțeles, va trebui să adresați o întrebare la care să se poată răspunde prin `DA` sau `NU`. Ați putea încerca: *Este su-*

ficientă o precizie de trei zecimale? Răspunsul la întrebare va fi ANUNȚ GENERAL și va fi urmat de publicarea în secțiunea dedicată problemelor (**www.ginfo.ro/concurs/probleme.shtml**) a unui mesaj de genul:

Rezultatul va trebui scris cu o precizie de două zecimale. Ne cerem scuze pentru descrierea incompletă a problemei.

Sperăm să nu fim nevoiți să publicăm prea des astfel de mesaje, dar... După citirea anunțului general puteți începe scrierea codului sursă. O soluție corectă ar putea fi:

```
 1: Program SUMA;
 2: var a,b:Real;
 3:     f:Text;
 4: Begin
 5:   Assign(f,'SUMA.IN');
 6:   Reset(f);
 7:   Readln(f,a,b);
 8:   Close(f);
 9:   Assign(f,'SUMA.OUT');
10:   Rewrite(f);
11:   Writeln(f,a+b:0:2);
12:   Close(f)
13: End.
```

Numerele de la începutul fiecărei linii nu fac parte din program; ele sunt folosite pentru a ne putea referi mai ușor la o anumită linie.

Dacă veți adăuga o linie `1': uses Crt;` și o linie `4': Clrscr;` nu veți reuși să obțineți decât eroarea `Error 200: Division by zero` așa cum am explicat în secțiunea *Sfaturi*.

Dacă adăugați o linie `12': Readln;` rezultatul evaluării va fi `Time out!`, deoarece execuția programului nu se va termina, ci se va aștepta apăsarea tastei **<ENTER>**. Execuția va fi întreruptă de către programul de evaluare după o secundă.

Dacă folosiți linia `5: Assign(f,'C:\SUMA.IN');` atunci programul va funcționa corect numai dacă directorul curent este rădăcina *drive*-ului `C:` (ceea ce este foarte improbabil). Nu veți obține decât eroarea `File not found`.

Dacă folosiți linia `5: Assign(f,'ALTFISIER');` atunci programul nu va citi datele de intrare din fișierul specificat și va rula incorect.

Dacă folosiți linia `9: Assign(f,'C:\SUMA.OUT');` atunci programul va părea că funcționează corect, dar programul de evaluare nu va găsi fișierul dumneavoastră de ieșire decât dacă directorul curent este rădăcina *drive*-ului `C:` (din nou, șansele sunt foarte mici). Programul de evaluare va returna `No output file!`.

Dacă folosiți linia `9: Assign(f,'ALTFISIER');` atunci programul nu va scrie datele în fișierul de ieșire specificat, iar programul de evaluare va returna `No output file!`.

Dacă folosiți linia `7: Readln(a,b);` atunci programul va aștepta datele de intrare de la intrarea standard și va fi întrerupt după o secundă. Puteți folosi această linie numai dacă aveți o secvență asemănătoare cu:

```
5: Assign(Input,'SUMA.IN');
6: Reset(Input);
7: Readln(a,b);
8: Close(Input);
```

Dacă folosiți linia `11: Writeln(a+b:0:2);` atunci datele de ieșire vor fi scrise pe ecran și nu în fișierul de ieșire specificat. Puteți folosi această linie numai dacă secvența este de tipul:

```
 9: Assign(Output,'SUMA.OUT');
10: Rewrite(Output);
11: Writeln(a+b:0:2);
12: Close(Output)
```

Acestea sunt cele mai des întâlnite erori care au apărut la precedentele ediții ale concursului organizat de *GInfo*. Sperăm că nu veți face aceleași greșeli și la această ediție.

După scrierea codului sursă trebuie să trimiteți soluția. În cele ce urmează vom considera că *ID*-ul dumneavoastră este `02457` și participați la prima rundă a concursului . Trebuie să creați arhiva pe care o veți trimite; aceasta se va numi **`02457R01.ZIP`**, **`02457R01.ACE`** sau **`02457R01.RAR`** și va conține două fișiere: **`02457R01.BAT`** și **`02457R01.PAS`**.

Fișierul **`02457R01.PAS`** va conține codul sursă al rezolvării, în timp ce fișierul **`02457R01.BAT`** va conține linia de comandă care va fi folosită pentru compilare, de exemplu `bpc 02457R01.PAS`.

Folosiți `bpc` pentru compilarea surselor *Pascal* și `bcc` pentru compilarea surselor *C/C++*.

După crearea acestui fișier trebuie construită arhiva pe care o veți trimite. Folosiți un arhivator care creează arhive `.RAR`, `.ACE` sau `.ZIP`; nu creați o arhivă autoexpandabilă. Acum puteți trimite *e-mail*-ul care va conține soluția. Nu uitați să atașați arhiva și să scrieți în linia *Subject*: `02457 Runda 01`.

Modificarea regulamentului

Redacția *GInfo* își rezervă dreptul de a modifica prezentul regulament pe parcursul desfășurării concursului.

Dacă apar prevederi noi sau dacă anumite prevederi ale regulamentului sunt modificate sau eliminate, atunci acestea vor fi anunțate pe *site*-ul concursului.

Orice modificare a regulamentului va intra în vigoare, în funcție de situație, fie din momentul publicării ei, fie după încheierea rundelor aflate în desfășurare. În momentul anunțării modificării va fi precizat și momentul din care aceasta intră în vigoare.

Întreruperea sau anularea concursului

Editura *Agora Media* și redacția *GInfo* își rezervă dreptul de a întrerupe desfășurarea concursului sau de a anula întregul concurs în orice moment dacă este cazul.

Metode de ABORDARE a problemelor NP

Mihai Stroe

Problemele nedeterminist polinomiale ridică dificultăți destul de mari majorității elevilor care participă la concursurile de programare. În cadrul acestui articol vom prezenta câteva modalități de abordare a acestora.

Problemele care nu se pot rezolva perfect prin algoritmi de complexitate polinomială aparțin clasei *NP \ P*. În cele ce urmează, vom folosi denumirea *NP* pentru problemele de acest tip. Majoritatea observațiilor se pot aplica și problemelor pentru care, deși există metode polinomiale de rezolvare, acestea nu sunt cunoscute de către rezolvitor (de exemplu, cazul problemelor mai dificile de la concursuri).

Domeniul problemelor *NP* este deosebit de amplu; ne vom limita prezentarea asupra câtorva aspecte interesante ale acestuia. Expunerea presupune cunoașterea metodei *backtracking*.

Încadrarea problemei în clasa NP \ P

Există câteva probleme standard pentru care nu se cunosc algoritmi de rezolvare polinomiali. În cazul în care întâlniți una dintre acestea, în principiu nu are rost să căutați un algoritm eficient de rezolvare.

Atenție, aparențele pot fi însă înșelătoare și ceea ce poate părea o problemă *NP* clasică poate fi de fapt rezolvabilă, datorită introducerii unor condiții suplimentare.

De exemplu, problema colorării unui graf cu *K* culori, prezentată mai jos, este de fapt *NP*, dar pentru *K*=2 există algoritmi polinomiali. De asemenea, problema submulțimii de sumă dată se rezolvă prin metoda programării dinamice, dacă valorile implicate sunt numere întregi relativ mici (de exemplu, suma este cel mult 10000) și prin alte metode eficiente, dacă renunțăm la condiția determinării exacte a sumei.

Dacă reușim să demonstrăm că, rezolvând problema dată, am rezolva una dintre problemele "clasice" *NP-complete*, atunci problema este *NP-completă*. Pe scurt, o problemă este *NP-completă* dacă nu există un algoritm polinomial pentru rezolvarea ei, dar rezultatul poate fi verificat în timp polinomial.

De exemplu, problema determinării tuturor submulțimilor unei mulțimi nu este *NP*-completă (deși este *NP*) deoarece numărul submulțimilor crește exponențial în funcție de numărul elementelor submulțimii, deci rezultatul nu poate fi verificat în timp polinomial. Câteva dintre cele mai cunoscute probleme *NP* sunt următoarele:

Problema satisfiabilității formulei logice

Se consideră o expresie în care pot apărea variabile logice (valorile pot fi 0 sau 1) și operatorii logici de conjuncție, disjuncție și negație. Să se atribuie valori variabilelor astfel încât rezultatul evaluării expresiei să fie 1.

Problema submulțimii de sumă dată

Se consideră o mulțime *M* cu *n* elemente (numere reale strict pozitive) și un număr *D*. Să se găsească o submulțime *S* a mulțimii *M*, astfel încât suma elementelor lui *S* să fie mai mică sau egală cu *D* și cât mai apropiată de acesta.

Problema acoperirii cu mulțimi (set covering)

Se consideră o mulțime *M* și o mulțime *P* de submulțimi ale lui *M*. Să se selecteze o submulțime *S* a lui *P*, de cardinal minim, astfel încât fiecare element din *M* să se afle în cel puțin una din mulțimile din *P*.

Problema colorării

Se consideră un graf neorientat și un număr *K*. Să se coloreze graful cu *K* culori (sau cu un număr minim de culori), astfel încât oricare două noduri adiacente să fie colorate diferit.

Problema clicii

Se consideră un graf neorientat. Să se aleagă o submulțime de noduri, de cardinal maxim, astfel încât între oricare două noduri să existe muchie.

Problema acoperirii cu vârfuri

Se consideră un graf neorientat cu *N* vârfuri și *M* muchii. Să se determine o submulțime minimală *N'* a vârfurilor grafului astfel încât toate cele *M* muchii să aibă cel puțin o

extremitate care face parte din mulțimea N' (cu alte cuvinte, vârfurile din submulțimea aleasă trebuie să "acopere" toate muchiile).

Problema ciclului hamiltonian

Se consideră un graf neorientat (în cazul abordat aici) și o funcție distanță definită pe mulțimea nodurilor grafului, cu valori strict pozitive. Să se găsească un ciclu care trece prin fiecare nod exact o dată și are costul minim (de fapt, se va aborda varianta "și are costul cât mai mic"). Costul ciclului este suma distanțelor date de muchiile pe care le conține.

Problema jocului Perspico

Aceasta nu este o problema *NP* "clasică", dar o vom folosi în expunerea care urmează.

Se consideră o matrice cu patru linii și patru coloane care conține toate numerele cuprinse între 0 și 15.

Să se obțină configurația:

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

folosind un număr minim de mutări. Mutările permise sunt interschimbări ale elementului 0 cu unul dintre elementele de pe pozițiile vecine pe orizontală și verticală.

Simplificarea problemei

De multe ori, problema poate fi simplificată înaintea aplicării algoritmului de rezolvare. După obținerea soluției pentru varianta simplificată, soluția pentru întreaga problemă se obține foarte rapid, printr-o metodă polinomială.

De exemplu, în cazul problemei colorării nodurilor unui graf cu K culori, există o simplificare dată de următoarea propoziție:

"*Problema pentru un graf G în care există un nod P cu gradul mai mic decât K se poate rezolva ușor dacă rezolvăm problema pentru graful $G \setminus \{P\}$.*"

Demonstrația este evidentă. Pentru a obține soluția pentru graful G se obține soluția pentru graful $G \setminus \{P\}$, după care se colorează nodul P cu o culoare care nu a fost atribuită nici unui vecin. Datorită faptului că numărul culorilor este mai mare decât gradul nodului P, colorarea este posibilă.

Astfel, pașii algoritmului de rezolvare sunt următorii:

- se elimină din graf toate nodurile cu gradul mai mic decât K;
- se repetă pasul anterior până când nu mai exista în graf noduri cu gradul mai mic decât K (după eliminările de la primul pas, gradele altor noduri pot deveni mai mici decât K);
- se rezolvă problema pentru graful rămas (și pentru problema particulară în care gradele tuturor nodurilor sunt cel puțin egale cu K);
- se construiește soluția problemei inițiale, colorând nodurile eliminate în ordinea inversă ordinii eliminării.

Alegerea unei modalități de rezolvare

Înaintea abordării unei probleme *NP* este necesară o analiză atentă. Anumite probleme se pretează mai bine anumitor tipuri de rezolvări.

Cele mai importante modalități de abordare pentru probleme *NP* sunt:

- explorarea în lățime a spațiului soluțiilor;
- explorarea în adâncime a spațiului soluțiilor;
- metode euristice;
- metode bazate pe nedeterminism;
- algoritmi genetici;
- algoritmi pseudo-polinomiali;
- scheme de aproximare;
- rețele neuronale.

Metodele de rezolvare pot fi combinate. De exemplu, înaintea aplicării *backtracking*-ului se poate căuta o soluție printr-o metodă euristică; rezultatul obținut va fi folosit pentru evitarea explorării unor cazuri neinteresante.

Câteva dintre avantajele și dezavantajele unor metode sunt descrise în continuare.

Eliminarea situațiilor neinteresante

Majoritatea abordărilor prezentate pot fi îmbunătățite folosindu-se o tehnică de eliminare a unor elemente din spațiul soluțiilor (***pruning***).

Eliminarea poate fi realizată euristic (se estimează, rapid, dacă elementul respectiv "promite" să conducă la o soluție bună și, dacă nu, este eliminat), dar există și probleme pentru care eliminarea este sigură.

De exemplu, în cazul unor probleme de minim, poate există o funcție care aproximează inferior costul transformării configurației curente în soluție a problemei. Această funcție este numită ***euristică optimistă***. Dacă suma dintre costul configurației și funcția aceasta este mai mare decât costul celei mai bune soluții obținute anterior în cadrul algoritmului, configurația curentă poate fi eliminată, deoarece nu va genera o soluție mai bună.

Cu cât funcția aproximează mai bine costul transformării, cu atât algoritmul este mai performant, deoarece elementele neperformante sunt excluse, împreună cu toate elementele (tot neperformante) care ar fi fost introduse de ele.

Pentru problema jocului *Perspico*, o funcție *euristică optimistă* bună este suma *distanțelor Manhattan* de la poziția fiecărui element din matrice până la poziția sa finală (fără a aduna și distanța pentru elementul 0). *Distanța Manhattan* dintre două puncte este suma dintre distanța pe orizontală și distanța pe verticală.

Pentru problema ciclului hamiltonian, cea mai bună *euristică optimistă* cunoscută se bazează pe calculul arborelui minim de acoperire pentru nodurile care încă nu au fost selectate. Se pot alege și alte funcții euristice pentru eliminarea unor configurații, de exemplu soluția dată de un *greedy* rapid (se construiește un drum alegând la fiecare pas muchia de cost minim), dar aceasta nu este *euristică optimistă*.

Alegerea ordinii explorării soluțiilor

Metodele bazate pe explorarea în lățime a grafului configurațiilor și cele bazate pe explorarea în adâncime utilizează deseori o anumită ordine în care sunt explorate soluțiile. Problema pe care se va baza expunerea este cea a jocului *Perspico*.

În cazul explorării în lățime, configurațiile se introduc într-o coadă. O aceeași configurație nu va fi introdusă de două ori. Pentru aceasta se folosesc structuri de date avansate care permit căutarea rapidă, cum ar fi tabelele de dispersie. Dacă numărul configurațiilor este redus, se poate folosi o structură de selecție simplă (o mulțime sau un vector de valori logice). Principalul dezavantaj constă în depășirea rapidă a capacității memoriei disponibile, dacă spațiul configurațiilor este mare.

În cazul explorării în adâncime, metoda cea mai des folosită este *backtracking*. Nu prezentăm pe larg metoda, deoarece o considerăm cunoscută.

Se folosește o stivă. La fiecare pas se introduce în stivă una dintre configurațiile vecine celei din vârful stivei; după ce căutarea pentru aceasta s-a terminat, se va introduce o altă configurație vecină etc. Memoria disponibilă nu este o problemă (structura de bază este o stivă de configurații), în schimb există posibilitatea explorării repetate a aceleiași stări. În plus, soluțiile foarte apropiate de configurația de plecare pot fi pierdute.

Există o metodă care înlătură acest ultim neajuns, depășind și limitarea de memorie dată de explorarea în lățime. Tehnica se numește ***Depth First Search with Iterative Deepening*** (pe scurt ***DFSID***) și constă în parcurgerea în adâncime a regiunilor din graful configurațiilor apropiate de configurația de start. Este explorată inițial mulțimea configurațiilor aflate la distanța 1, apoi la distanța 2, 3 etc. de configurația inițială (aici prin "distanță" înțelegem lungimea drumului minim în graful configurațiilor; pentru jocul *Perspico* aceasta reprezintă numărul de mutări efectuate). Algoritmul se încheie la găsirea unei soluții bune sau la depășirea timpului acordat. Evident că *DFSID* duce la explorarea repetată a unor noduri ale grafului, dar acesta nu este un neajuns foarte mare, comparativ cu avantajele aduse.

De exemplu, dacă la fiecare introducere în stivă ar exista exact trei opțiuni, atunci *DFSID* ar genera, la pasul K, $3^1 + 3^2 + 3^3 + \ldots + 3^K$ configurații. Numărul de configurații generate la toți pașii precedenți este net inferior (demonstrația se face prin inducție și este foarte simplă), deci performanța, din punct de vedere al vitezei, scade de mai puțin de două ori față de cea a explorării în lățime.

Tehnica *DFSID* se referă la modificarea adâncimii stivei în *backtracking*, dar există și alte variante ale acestui stil de abordare. De exemplu, în problema ciclului hamiltonian se poate încerca o abordare în care nu se acceptă soluții cu un cost mai mare decât K. Dacă nu este găsită nici o soluție, se procedează la incrementarea lui K cu un anumit pas și se reia explorarea.

Toate aceste abordări pot fi îmbunătățite prin tehnicile de *pruning* amintite anterior. Alegerea unei metode trebuie să fie, de obicei, completată de găsirea unor tehnici de eliminare a nodurilor neinteresante.

Există o variantă mai bună a explorării în lățime, și anume algoritmul *A**, folosit pentru problemele de minim. În cazul acestui algoritm, se folosește tehnica de *pruning* bazată pe *euristică optimistă*.

La fiecare pas, sunt introduși în coadă succesorii celei mai promițătoare configurații. Astfel, pentru fiecare configurație se calculează suma dintre costul ei și *euristica optimistă*. Configurația cu această sumă minimă va fi expandată. Algoritmul garantează obținerea soluției optime; sunt necesare structuri de date care permit găsirea eficientă a minimului.

Pentru problema jocului *Perspico* se va folosi *euristica optimistă* descrisă anterior. Se observă că ea poate fi calculată foarte ușor la trecerea de la o configurație la alta (practic se schimbă distanța unui singur element din matrice față de poziția sa finală).

În general, un program care implementează *A** este mult mai complex decât unul care implementează *DFSID*. În plus, memoria necesară este exponențială pe cazurile defavorabile. De aceea, cu toate că explorează mai multe configurații, *DFSID* combinat cu *pruning* este mai indicat în multe cazuri. Avantajele *DFSID* au fost tratate pe larg în articolul ***Branch&Bound și backtracking***, publicat de *Ovidiu Gheorghioiu* în *GInfo* 9/2 (februarie 1999).

Particularitățile problemelor

În multe cazuri, particularitățile problemelor pot permite optimizări interesante. Vom examina modalități de abordare pentru două probleme *NP*.

Recomandăm cititorului ca, după lecturarea enunțului problemelor, să întrerupă citirea acestui articol și să se gândească puțin la modalitățile de rezolvare și la posibilele optimizări. Observațiile pot fi apoi comparate cu cele ale autorului. Deși nu toate observațiile autorului sunt implementabile în timp de concurs, ele funcționează bine pentru seturi de date de intrare mari.

Probleme

Se consideră un graf neorientat cu cel mult 100 de noduri și 1000 de muchii. Graful se citește dintr-un fișier; pe prima linie se află numărul de noduri N și numărul de muchii M, după care urmează cele M muchii, câte una pe linie. Toate nodurile au gradul cel puțin 1.

1) Să se selecteze o submulțime S a mulțimii nodurilor, de cardinal minim, astfel încât fiecare nod care nu este în S să aibă cel puțin un vecin în S (din mulțimea de noduri S să se "vadă" toate nodurile).
2) Să se selecteze o submulțime S a mulțimii nodurilor, de cardinal minim, astfel încât fiecare muchie din graf să aibă cel puțin unul din nodurile incidente în S (din mulțimea de noduri S să se "vadă" toate muchiile).

Fiecare mulțime S se va afișa prin numerele de ordine ale elementelor. Timpul de execuție nu trebuie să depășească o secundă.

În continuare vom prezenta modul de rezolvare al celor două cerințe.

În majoritatea cazurilor, problema 2) este mai simplă decât 1). Se observă că, dacă un nod X nu se află în S, toți vecinii lui se vor afla în mod obligatoriu în S. În caz contrar, o parte din muchiile incidente lui X nu vor fi "văzute" din S.

Această observație conduce la o soluție *backtracking*. Nodurile sunt procesate în ordinea descrescătoare a gradelor; la o iterație este introdus în stivă fie nodul curent (ceea ce elimină multe muchii introducând un singur nod în S), fie toți vecinii săi (operație care micșorează mult graful, dar introduce noduri suplimentare în S). Dacă numărul de noduri din S este mai mare decât un optim găsit anterior, căutarea pe varianta curentă este abandonată.

Această abordare poate fi mult îmbunătățită. De exemplu, nodurile de grad 1 nu vor fi selectate în soluția optimă (sau, dacă ar fi, o soluție la fel de bună se obține înlocuindu-le cu vecinii lor), deci pot fi eliminate din graf înainte de începerea căutării și nodurile adiacente lor vor fi introduse în soluție (ceea ce duce la eliminarea unor muchii "văzute" de aceste noduri și eventual la apariția unor alte noduri de grad 1 etc.). Excepție fac perechile de noduri de grad 1, caz în care un nod din fiecare pereche este eliminat, iar celălalt este selectat automat.

Alte posibile optimizări:

- la un moment dat, un nod care are toți vecinii în S (introduși anterior) nu va fi introdus în S pentru că nu aduce nimic în plus, în schimb conduce la o soluție mai slabă;
- dacă graful nu este conex, problema se va rezolva separat pentru fiecare componentă conexă; optimizarea se poate aplica și pe parcurs (în momentul deconectării grafului prin eliminarea unor noduri și muchii);
- se poate folosi o *euristică optimistă* pentru eliminarea unor variante. De exemplu, dacă în graf au mai rămas N_1 noduri și M_1 muchii, este necesar ca suma gradelor nodurilor care vor fi introduse în soluție să fie cel puțin M_1. Se aleg din cele N_1 noduri rămase nodurile cu gradele cele mai mari (relativ la cele M_1 muchii), până când suma gradelor ajunge să fie cel puțin M_1 (chiar dacă aceste noduri nu "văd" cele M_1 muchii). Dacă numărul de noduri introduse, plus numărul de noduri existente, depășesc optimul obținut anterior, varianta curentă trebuie abandonată.

Presupunând că nu se mai introduc optimizări în căutare, pasul următor îl constituie scrierea unui program care să execute operațiile descrise. Optimizările de cod contează foarte mult, deoarece vor fi examinate mai multe variante și șansele de a se obține o soluție mai bună cresc.

Problema 1) este un caz particular al problemei acoperirii cu mulțimi (*set covering*), cunoscută ca fiind *NP-completă*. Astfel, o mulțime este formată dintr-un nod și din vecinii lui. Acest fapt nu este suficient pentru a demonstra că 1) este *NP*, dar în cazul de față problema 1) este într-adevăr *NP*. Mai mult, particularitățile problemei 1) fac ca anumite optimizări cunoscute pentru *set covering* să funcționeze foarte bine. Din aceste motive putem trata 1) printr-o abordare care rezolvă problema mai generală. Este de reținut faptul că o astfel de abordate (rezolvarea unei probleme mai generale) nu este recomandată în majoritatea cazurilor, deoarece se pierde informația specifică problemei.

În continuare vom discuta problema acoperirii cu mulțimi. Începem cu două observații fundamentale:

- dacă o mulțime este inclusă în alta, ea nu va intra în soluția finală;
- dacă un element din mulțimea $\{1, 2, ..., N\}$ este conținut într-o singură mulțime, aceasta va face parte din soluția finală.

În plus, intuitiv este avantajos să dăm șanse mai mari mulțimilor cu multe elemente să facă parte din soluția finală. Din nefericire, algoritmul *greedy* care ar rezulta din această ultimă observație nu dă întotdeauna soluția optimă.

Observațiile conduc la o rezolvare *backtracking*, care poate fi implementată recursiv. Aceasta funcționează astfel:

- pasul "elimină": se elimină toate mulțimile care sunt incluse în alte mulțimi (atenție la cazul cu mulțimi egale);
- pasul "găsește": se caută un element conținut într-o singură mulțime; dacă un astfel de element este găsit, mulțimea respectivă este selectată, atunci se continuă cu apelarea *back(k+1)* și se iese din procedura recursivă;
- dacă pasul "găsește" eșuează, se va apela *back(k+1)* selectând, pe rând, fiecare mulțime rămasă, în ordinea descrescătoare a numărului de elemente;
- la apelul *back(k+1)* se va lucra cu mulțimile rămase, din care se elimină toate elementele care se găsesc în mulțimi deja selectate (ceea ce poate duce la succese noi ale pașilor "elimină" și "găsește").

Observație

Utilitatea abordării problemelor *NP* nu poate fi pusă la îndoială. O soluție bună pentru problema 1) poate duce, de exemplu, la scăderea costurilor necesare pentru deschiderea unei rețele de *fast food*-uri care să acopere integral piața într-o anumită zonă etc.

Concluzii

Problemele *NP* apar în numeroase domenii. Deși se știe că timpul necesar pentru rezolvare este exponențial, aceasta nu înseamnă că studiul lor ar trebui abandonat. Alternativa este de a găsi soluții practice cât mai eficiente.

Pentru abordarea cu succes a unei probleme *NP* este necesară o analiză atentă, cunoașterea tuturor opțiunilor și alegerea celor mai bune soluții. Aceasta poate implica testări complexe ale mai multor programe, folosind date de test cât mai apropiate de cele care vor apărea pe parcursul utilizării soluției finale.

Mihai Stroe este student în anul V la Universitatea Politehnica din București și poate fi contactat prin e-mail la adresa mihai_stroe@yahoo.com.

Drumuri MINIME când costurile sunt MICI

Mircea Digulescu, Andrei Matei

Problema determinării celui mai scurt drum de la un nod la altul într-un graf este una fundamentală în algoritmică. În continuare vom prezenta un algoritm al cărui timp de execuție depinde de costurile asociate muchiilor. Acesta este aplicabil atunci când costurile sunt mici și sunt numere întregi nenegative.

Notații

Considerăm un graf $G = (V, E)$ și o funcție de cost $w: E \to N$ (astfel încât $w(e) \geq 0, \forall e \in E$) care asociază un întreg nenegativ fiecărei muchii a grafului. Costul unui drum p în G este suma costurilor tuturor muchiilor din p. Fie nodul sursă s.

Definim $\delta(u,v) = \min(Cost(u \xrightarrow{p} v))$ ca fiind costul minim dintre toate căile p de la u la v, unde $u, v \in V$. De asemenea, fie $C = \max(w(e))$ costul celei mai "scumpe" muchii din graf și fie $C^* = \max(\delta(s, u))$ cel mai mare cost al unui drum minim în G care pleacă din s.

Pentru toți $v \in V$, $d[v]$ reprezintă *o estimare a costului unui drum minim* ($d[v] \geq \delta(s, v)$) pe tot parcursul execuției algoritmilor prezentați aici).

Obiectivul este găsirea costurilor $\delta(s, v)$ pentru orice $v \in V$. De asemenea, un drum p de cost minim de la s la v trebuie găsit, dar algoritmul folosit pentru determinarea $\delta(s, v)$ este simplu de modificat pentru a găsi un astfel de drum p.

Algoritmul prezentat va folosi liste dublu-înlănțuite. Pentru o listă dublu-înlănțuită l operația $v \Leftarrow l$ extrage primul element. După extracție, v va avea valoarea primului element din listă. Operația $l \Leftarrow v$ introduce un element v în lista l. Totodată, operația l a v are ca efect ștergerea elementului v din lista l. Valoarea din v rămâne neschimbată în urma aceastei operații.

Un algoritm de ordinul O(V+E+C*)

În continuare vom prezenta un algoritm care determină costul drumului minim atunci când muchiile au costuri întregi nenegative. El reprezintă, de fapt, o modificare a algoritmului lui *Dijsktra* în care este folosită o coadă de priorități neobișnuită.

Fie $l[0]$, $l[1]$, ..., $l[C^*]$ și $l[\infty]$ liste dublu-înlănțuite de noduri din V.

În total avem $C^* + 2$ liste. Pe parcursul execuției algoritmului, fiecare nod va aparține unei singure liste. Pentru un nod v, $d[v]$ va fi o estimare a costului drumului minim la v. Când algoritmul se încheie, vom avea $d[v] = \delta(s, v)$ pentru toate nodurile $v \in V$. Algoritmul poate fi ușor modificat pentru a determina și un drum de cost minim și lăsăm acest lucru pe seama cititorului.

În continuare vom prezenta varianta în pseudocod a algoritmului:

```
 1: l[i] ← ∅, 0 ≤ i ≤ C*
 2: l[∞] ← ∅
 3: l[0] ⇐ s
 4: pentru v ∈ V execută
 5:    l[∞] ⇐ v
 6:    d[v] ← ∞
 7: sfârșit pentru
 8: d[s] ← 0
 9: p ← 0
10: cât timp p < C* execută
11:    cât timp l[p] ≠ ∅ execută
12:       v ⇐ l[p]
13:       pentru (v, u) cu d[u] > d[v] + w(v, u)
          execută
14:          l[d[u]] a v
15:          d[u] ← d[v] + w(v, u)
16:          l[d[u] ⇐ u
17:       sfârșit pentru
18:    sfârșit cât timp
19: p ← p + 1
20: sfârșit cât timp
```

Algoritmul anterior este asemănător cu algoritmul lui *Dijsktra* prin faptul că procesează nodurile în ordine crescătoare a costului căii minime la ele. Liniile 1-8 realizează inițializarea valorilor l, S și d. Pe parcursul execuției algoritmului lista $l[i]$ va conține toate nodurile v pentru care $d[v] = i$. Algoritmul parcurge în ordine $l[0]$, $l[1]$, $l[2]$, ... și la pasul p relaxează toate muchiile adiacente cu un nod din $l[p]$. Liniile 14 și 16 vor muta un nod u pentru care estimarea costului drumului minim până la el, $d[u]$, s-a schimbat (ca urmare a relaxării unei muchii) în lista corespunzătoare noii estimări (mai întâi este șters din lista veche și apoi inserat în lista corespunzătoare).

Corectitudinea algoritmului

Vom arata că atunci când algoritmul se încheie, vom avea $d[v] = \delta(s, v)$ pentru orice $v \in V$. Vom demonstra mai întâi câteva rezultate intermediare.

Lema 1: *Dacă un nod v este extras în linia 12, atunci toate nodurile u care sunt legate de v (direct sau indirect) prin muchii de cost 0 vor fi procesate la aceeași iterație (a ciclului care începe la linia 10), ca și v, și vor avea $d[u] = d[v]$.*

Demonstrația se realizează prin inducție după numărul de muchii de cost 0 dintre u și v. Toate nodurile adiacente cu v vor fi descoperite de codul din linia 13, vor avea $d[u] = d[v]$ datorită liniei 15 și vor fi adăugate în lista curentă la linia 16. Prin inducție, dacă nodurile aflate la distanța k de v au fost procesate, atunci nodurile aflate la distanța $k + 1$ de v vor fi descoperite de codul din linia 13 și procesate. Deci, toate nodurile legate de v prin muchii de cost 0 vor fi procesate și vor avea $d[u] = d[v]$.

Lema 2: *La sfârșitul fiecărei iterații a ciclului care începe la linia 10 toate nodurile v pentru care $\delta(s, v) \le p$ au fost procesate.*

Demonstrația se realizează tot prin inducție. Primul nod extras este s. Din moment ce costurile muchiilor sunt nenegative, singurele noduri pentru care $\delta(s, v) \le 0$ sunt cele pentru care $\delta(s, v) = 0$. Orice nod pentru care $\delta(s, v) = 0$ trebuie să fie conectat de sursă prin muchii de cost 0. Conform lemei 1, toate nodurile de acest tip vor fi procesate.

La pasul p, datorită ipotezei de inducție, toate nodurile x pentru care $\delta(s, v) < p$ au fost procesate. Fie v un nod astfel încât $\delta(s, v) = p$. Fie u predecesorul direct al lui v pe un drum de cost minim s:

$$\delta(s, v) = \delta(s, u) + w(u, v) \Rightarrow p = \delta(s, u) + w(u, v).$$

Dacă $w(u, v) > 0$, atunci $\delta(s, u) < p$, deci u a fost procesat la o iterație precedentă. Deoarece u a fost procesat atunci când muchia (u, v) a fost relaxată, nodul v a fost inserat în lista $l[p]$ (dacă nu era inserat deja în listă). Deoarece toate nodurile din $l[p]$ sunt procesate la pasul p, atunci nodul v va fi procesat.

Cazul în care $w(u, v) = 0$ nu prezintă interes, deoarece, conform lemei 1, toate nodurile legate prin muchii de cost 0 de nodul u sunt procesate la aceeași iterație ca și u. Deci, dacă am dovedit că toate nodurile pentru care $\delta(s, v) = p$ și care sunt legate de predecesorii direcți prin muchii de cost pozitiv sunt procesați la pasul p, atunci toate nodurile pentru care $\delta(s, v) = p$ sunt procesate la pasul p.

Teoremă: *Atunci când un nod este extras în linia* 12 *el va avea $d[v] = \delta(s, v)$.*

Demonstrația se realizează tot prin inducție. Primul nod extras este s și va avea $d[s] = \delta(s, s) = 0$.

Fie v un nod arbitrar extras din $l[p]$ cu operația de la linia 12. Prin inducție avem calculate costurile drumurilor minime pentru toate nodurile care au fost extrase înaintea lui. Datorită lemei 2 toate nodurile pentru care costul drumului minim este mai mic decât p au fost procesate. Astfel avem: $d[u] = \delta(s, u)$, pentru toți u astfel încât $\delta(s, u) < p$.

Fie un drum de cost minim de la s la v: $s \xrightarrow{*} u \to v$. Vom avea $\delta(s, v) = \delta(s, u) + w(u, v)$.

Presupunem, fără a reduce generalitatea, că $w(u, v) \ge 0$ (deoarece, conform lemei 1, toate nodurile care sunt conectate prin muchii de cost 0 de un nod x vor avea în cele din urmă $d[u] = d[x]$ care va fi $\delta(s, u)$ dacă $d[x] = \delta(s, x)$). Prin urmare: $\delta(s, u) < \delta(s, v) \Rightarrow \delta(s, u) < p$.

Conform lemei 2, toate nodurile de acest tip au fost procesate. Fie u primul asemenea nod procesat de algoritm. Prin ipoteza de inducție avem $d[u] = \delta(s, u)$. Atunci când muchia (u, v) este relaxată avem $d[v] = \delta(s, u) + w(u, v)$ și, datorită alegerii făcute, pentru u (ca predecesor al lui v pe un drum de cost minim), vom avea $d[v] = \delta(s, v)$.

În sfârșit, deoarece fiecare nod aparține unei liste (prin inițializare), și toate listele sunt procesate, toate nodurile vor fi procesate. Astfel, conform teoremei 1, vom avea $d[v] = \delta(s, v)$ pentru toți $v \in V$ când algoritmul se încheie.

Analiza complexității

Linia 1 se execută într-un timp de ordinul $O(C^*)$; ordinul de complexitate al operațiilor efectuate în liniile 4-7 este $O(V)$, iar cel al operațiilor efectuate în liniile 2, 3, 8 și 9 este $O(1)$.

Ciclul care începe la linia 10 necesită un timp $O(C^*)$, dacă nu luăm în considerare ciclul care începe la linia 11.

Se observă că dacă un nod este extras în linia 12 (și procesat) el nu va mai fi procesat ulterior. Prin urmare, timpul total consumat de execuțiile liniei 12 este $O(V)$, deoarece fiecare nod este extras exact o dată. Așadar, timpul necesar pentru toate execuțiile ciclului care începe la linia 11 este $O(V)$, dacă facem abstracție de timpul necesar execuției ciclului care începe la linia 13.

Se observă că fiecare muchie (v, u) a grafului este luată în considerare exact o dată în cadrul ciclului care începe la linia 13 (sau de două ori dacă graful este neorientat), mai exact atunci când nodul v este procesat (dacă graful este neorientat este considerată și atunci când este procesat nodul u; condiția $d[u] > d[v] + w(v, u)$ va fi adevărată cel mult o dată: la prima procesare a unui nod adiacent acesteia). Operațiile de inserare și eliminare pentru liste dublu-înlănțuite pot fi executate în timp constant; așadar ordinul de complexitate al tuturor execuțiilor acestui ciclu este $O(E)$.

În concluzie, ordinul de complexitate al algoritmului este $O(C^*) + O(V) + O(C^*) + (O(V) + O(E)) = O(V + E + C^*)$.

Datorită faptului că $C^* < V \cdot C$ ordinul de complexitate este cu siguranță mai mic decât $O(V \cdot C + E)$.

Bibliografie

1. T. H. Cormen, C. E. Leiserson, R. R. Rivest, *Introducere în algoritmi*, Computer Libris Agora, 2000
2. Ahuja, Mehlhorn, Orlin, Tarjan, *Faster algorithms for the shortest path problem*, Tehnical Report 193, MIT Operations Research Center, 1998
3. Gabow, Tarjan, *Faster scaling algorithms for network problems*, SIAM Journal on Computing, 1989

Mircea Digulescu și Andrei Matei sunt elevi în clasa a XI-a la Colegiul Național de Informatică Tudor Vianu din București și pot fi contactați prin e-mail la **mircea85@yahoo.com**, *respectiv* **andreimatei@home.ro**.

Conceptul de ALGORITM - abordare MODERNĂ

Marin Vlada

În cadrul acestui nou articol, dedicat unei abordări moderne a conceptului de algoritm, vom prezenta detalii referitoarea la elaborarea și reprezentarea algoritmilor. Vom prezenta modul în care pot fi descriși algoritmii, folosind limbajul pseudocod.

Evoluția *sistemelor de calcul* (*SC*) și a *limbajelor de programare* (*LP*) a determinat varietate și evoluție în *descrierea și elaborarea algoritmilor*. De altfel, reprezentarea algoritmilor și limbajele de programare s-au influențat reciproc.

Până la apariția și răspândirea *structurilor de control* (anii '70), reprezentarea algoritmilor s-a realizat prin ***limbajul schemelor logice*** și "***prin pași***" care necesitau utilizarea *instrucțiunii de salt* (*go to <etichetă>*).

Sintaxa și semantica limbajelor de programare din acea perioadă se supuneau acestei restricții (apariția și dezvoltarea unui număr mare de limbaje de programare - la un moment dat existau peste 200 de limbaje de programare - se datorează și căutării eliminării salturilor din procesele de calcul).

Cercetările, experiențele și implementările au durat peste 20 de ani. ***Algoritmica*** (*reprezentarea și elaborarea algoritmilor*) și ***programarea*** (*elaborarea și implementarea programelor*) au fost nevoite să creeze un "*front comun*" pentru implementarea structurilor de control și construirea primelor limbaje de programare moderne: *limbajul Pascal* (1972, *Niklaus Wirth*, *Universitatea din Zurich*) și *limbajul C* (1972, *Dennis Ritchie*, *AT&T Bell Laboratories*).

Apariția limbajelor *Pascal* și *C* au impus limbajul *pseudocod* în domeniul reprezentării și elaborării algoritmilor.

Mai târziu, evoluția rețelelor de calculatoare și dezvoltarea rețelei *Internet*, precum și implementarea unei versiuni perfecționate a limbajului *C* (versiunea *C++*), au condus la apariția și utilizarea pe scară largă a limbajului de programare *Java*.

Limbajul pseudocod

Limbajul *pseudocod* are o sintaxă și semantică asemănătoare limbajelor de programare moderne, având o anumită *flexibilitate* în ceea ce privește sintaxa, în ideea că, prin codificarea unui algoritm într-un limbaj de programare, operația să fie cât mai comodă. De aici, iese în evidență dependența reciprocă dintre *algoritmică* și *programare*, ambele fiind influențate de sistemele de calcul și de sistemele de operare actuale.

Un *algoritm* reprezentat în limbajul *pseudocod* este constituit dintr-o secțiune în care se declară variabilele și tipul de date asociat acestora, precum și definirea procedurilor sau funcțiilor apelate de algoritm, și corpul algoritmului, care este o succesiune finită de *instrucțiuni executabile*:

```
algorithm <nume_algoritm>
   <declarare_variabile>   // tipul variabilelor
   <declarare_rutine>      // definirea de proceduri / funcții
begin
   <procesul_de_calcul_P>  // instrucțiunile algoritmului
end
```

Din definiția dată conceptului de algoritm se poate observa că *structura* acestuia conține următoarele *elemente de bază*:

- ***date*** - *variabile și tipuri de date* utilizate pentru accesul la memorie și generarea de valori conform calculelor implementate în procesul de calcul prin intermediul instrucțiunilor;

- ***expresii*** - *formule de calcul* asemănătoare expresiilor matematice utilizate pentru *calcule aritmetice*, *logice* (*booleene*), operații asupra valorilor de tip *caracter* sau *șir de caractere*;
- ***instrucțiuni*** - *instrucțiuni sau comenzi executabile* pentru operații de *intrare / ieșire* și operații de prelucrare a datelor din memorie conform procesului de calcul;
- ***proceduri / funcții*** - *subprocese de calcul* cu o structură asemănătoare unui algoritm care pot fi executate prin așanumitele *instrucțiuni de apelare*.

Elementele lexicale ale limbajului sunt următoarele:

- ***identificatori*** - secvențe de caractere folosite pentru definirea *numelor variabilelor* și a *numelor procedurilor și funcțiilor*;
- ***expresii*** - forme lexicale asemănătoare expresiilor matematice care se construiesc folosind operanzi (constante, nume de variabile, apel de funcții), operatori (operații) și, eventual, paranteze pentru definirea priorităților operațiilor;
- ***cuvinte cheie*** - cuvinte din limba engleză care identifică un *tip de date* sau descriu o *instrucțiune*.

Date

Eficiența unui algoritm depinde atât de metodele și tehnicile implementate în procesul de calcul, cât și de tipurile de date utilizate. *Complexitatea algoritmilor* implică o complexitate atât a *metodelor utilizate*, cât și a *organizării datelor* în vederea prelucrării lor. De modul în care sunt *structurate datele* depinde eficiența algoritmilor.

O ***structură de date*** *este o colecție de date înzestrată cu o structură care precizează componentele și procedeele de reprezentare, identificare și înregistrare ale acestora.*

Limbajele de programare moderne oferă o mare varietate de *tipuri de date*, începutul l-a realizat limbajul *Pascal* care ulterior a implementat și tehnica OOP. Un *tip de date* este sistemul $T = (D, O)$, unde D este *domeniul de valori*, iar O este *mulțimea operatorilor / operațiilor* care acționează asupra valorilor din D.

Tipurile de date - care utilizează static memoria unui sistem de calcul - sunt clasificate astfel:

- ***simple / elementare*** - *data / informația* apare ca o entitate indivizibilă atât din punct de vedere al valorii, cât și în raport cu *unitatea centrală de prelucrare* (*UCP*); *o dată simplă / elementară* este specificată prin: *identificator*, *atribute* (domeniul de valori, modul de reprezentare în sistemul de calcul, precizia reprezentării), *valori* (enumerate sau indicate printr-o proprietate comună); din punct de vedere al domeniului de valori asociat, se disting următoarele *clase de date simple*: *integer* (numere întregi), *real* sau *float* (numere raționale), *boolean* (valori logice: *true*, *false*), *char* (caractere *ASCII*: cifre, litere, semne, simboluri etc.);
- ***structurate*** - *data / informația* este organizată într-o *structură* descrisă prin componente de același tip sau de tipuri diferite; componentele pot fi de tip *simplu / elementar* sau pot fi, la rândul lor, structuri; componentele vor fi identificate prin *nume* sau prin *poziția* pe care o ocupă în structură; din punct de vedere al structurii, se disting următoarele *clase de structuri*: *string* (*șir de caractere ASCII*: concatenarea datelor de tip *char*), *array* (*tablouri n-dimensionale*: structură cu componente de același tip care ocupă locații succesive în memoria sistemului de calcul, identificate prin poziție), *record* (*înregistrare*: structură cu componente de diverse tipuri, identificate prin nume); *file* (*fișier*: structură cu componente de tip *record*).

Tipurile de date prezentate mai sus utilizează memoria sistemului de calcul în *mod static*, nu întotdeauna eficient pentru algoritmul în cauză. Pentru utilizarea *dinamică* a memoriei sistemului de calcul, în scopul utilizării ei eficiente, este definită o organizare specială a datelor: *liste liniare* și *liste circulare*.

În același scop unele limbaje de programare moderne oferă tipul *referință* (*pointer*). Evident, în spiritul celor de până acum, conceperea și elaborarea unui algoritm trebuie să țină seama în detaliu de limbajul de programare utilizat, doar în final, pentru codificarea algoritmului.

O ***listă liniară*** este o structură de date omogenă, secvențială, formată din elementele unei mulțimi de date. O listă poate fi *vidă* (nu conține nici un element), sau *plină* (nu mai poate stoca alte elemente). Orice *listă liniară* are două elemente speciale, unul de început (baza listei) și altul de sfârșit (vârful listei).

Operațiile asupra elementelor unei liste sunt exprimate de *extragerea / accesarea* unui element din listă, *inserarea / adăugarea* unui element în listă, precum *și eliminarea / ștergerea* unui element din listă. Aceste operații vor fi implementate de utilizatori prin proceduri / funcții particulare, elaborate special pentru un anumit limbaj de programare (*Pascal*, *C++* etc.).

Stocarea listelor în memoria unui sistem de calcul se poate realiza în două moduri:

- ***secvențial*** - stocarea elementelor listei se face în *locații succesive* de memorie conform ordinii elementelor din listă; pentru a putea utiliza lista se reține primul element al listei (adresa de bază) care este baza listei; structura elementară adecvată reprezentării secvențiale a listelor este *tabloul unidimensional*;
- ***înlănțuit*** (*simplu sau dublu înlănțuit*) - lista înlănțuită este organizată secvențial, elementele fiind *dispersate în memorie*; fiecare element este înlocuit cu o celulă (nod) formată dintr-o *componentă de informație* (corespunzătoare elementului listei) și o *componentă de legătură*; în cazul listei simplu înlănțuite, partea de legătură conține *adresa celulei următorului element* din listă și se va reține adresa de bază a listei, iar ultima celulă va indica o valoare specială (*NIL*) care nu poate desemna o legătură; în cazul listei dublu înlănțuite, partea de legătură are o componentă care indică succesorul și o componentă care indică predecesorul.

Tipurile de date care utilizează dinamic memoria unui sistem de calcul sunt următoarele:

- ***stack*** (*stiva*) - listă liniară la care inserarea și extragerea elementelor se face prin vârful listei (de exemplu, stiva se utilizează pentru parcurgerea în adâncime (*DF - Depth First*) a arborilor / grafurilor, pentru generarea soluțiilor unei probleme folosind metoda *Backtracking* etc.);
- ***queue*** (*coada*) - listă liniară la care inserările se fac la baza listei, iar extragerile se fac prin vârful listei (de exemplu, *coada* se utilizează pentru parcurgerea pe lățime (*BF - Breath First*) a arborilor / grafurilor).

Listele liniare pot fi transformate în *liste circulare* dacă se consideră că *legătura ultimului element* indică *adresa bazei listei*. Utilizarea înlănțuirii permite o ușoară manipulare a listelor circulare prin considerarea unui singur *nod fictiv*, numit *bază*.

Declararea variabilelor

O ***declarare*** este constituită dintr-o parte în care se indică *tipul de date* - prin cuvintele cheie corespunzătoare - și o parte care reprezintă o *listă de nume de variabile* - prin identificatori corespunzători:

```
<tip_date> <listă_identificatori>.
```

Un ***identificator*** (*nume*) de variabilă reprezintă o secvență de caractere *ASCII* (maxim 256 caractere), primul caracter fiind *literă* (pentru unele limbaje de programare poate fi și caracterul "_"), iar celelalte fiind *litere*, *cifre* sau caracterul "_". Evident, lungimea secvenței de caractere trebuie să fie rezonabilă, suficient de mică pentru ca semantic să sugereze un aspect privind funcții, operații, metode etc. implementate în procesul de calcul.

Există libertate și flexibilitate mare în definirea identificatorilor pentru variabilele utilizate în conceperea și reprezentarea unui algoritm.

Exemple

```
integer a, b, c, max_X, x(100);
real x, y, z, m_produs(20,20);
boolean ind_1, ind_2;
char ch1, ch2;
string s1, s2, s3;
stack S;
queue Q1, Q2;
record(integer cod; string nume; real media)
                                    x, y, z, t;
```

Din punct de vedere sintactic, instrucțiunile sunt delimitate prin caracterul " ; " și pot fi scrise pe rânduri separate sau pe același rând.

Proceduri și funcții

Rezolvarea multor probleme complexe necesită executarea repetată a acelorași calcule / operații pentru date de intrare diferite.

Procedurile și *funcțiile* (în domeniul limbajelor de programare se numesc *subprograme*) oferă posibilitatea definirii și descrierii acestor calcule o singură dată. Prin intermediul subprogramului aceste calcule se execută prin apelarea lor ori de câte ori este necesar.

Orice subprogram este *identificat* printr-un *nume / identificator de procedură / funcție* și o *listă de parametri* (*argumente*).

O procedură poate fi declarată astfel:

```
Procedure <nume>[(<ListăParamFormali>)]
  [<DeclarareVariabile>] // tipul variabilelor locale
  [<DeclarareProceduri>] // definirea de proceduri / funcții
begin
  <ProcesulDeCalcul>     // corpul / instrucțiunile procedurii
end
```

Declararea unei funcții:

```
Function <nume> [(<ListăParamFormali>)]:
                                    <tip_rezultat>
  [<DeclarareVariabile>] // tipul variabilelor locale
  [<DeclarareProceduri>] // definirea de proceduri / funcții
begin
  <ProcesulDeCalcul>     // corpul / instrucțiunile funcției
end
```

Un algoritm este complet elaborat dacă este descrisă și definiția procedurilor și funcțiilor folosite. Definiția presupune descrierea instrucțiunilor pentru efectuarea calculelor proprii și modul cum se transmit *rezultatele* între programul principal (*programul apelant*) și *subprogramul apelat*.

În cazul unei proceduri, `<ListăParamFormali>` conține atât *parametri de intrare*, cât și *parametri de ieșire* (în urma apelării procedurii, vor stoca rezultatele calculate de instrucțiunile procedurii).

În cazul unei funcții, `<ListăParamFormali>` conține doar *parametri de intrare*, în urma apelării funcției, rezultatul va fi stocat în identificatorul (numele) funcției. De aceea este obligatoriu ca printre instrucțiunile funcției să existe o *instrucțiune de atribuire* în care membrul stâng să fie chiar identificatorul funcției.

Această deosebire dintre procedură și funcție face ca modul de apelare a acestora să fie diferit:

- ***apelarea unei proceduri*** este realizată prin instrucțiunea specială "*apel de procedură*" care are forma:

  `<nume>[<ListăParamActuali>];`

  în momentul în care este necesară execuția unei instrucțiuni de *apel de procedură*, se realizează o *corespondență* între parametrii formali (argumente) și parametrii ac-

tuali și se execută toate instrucțiunile specificate în definiția procedurii; după execuția ultimei instrucțiuni a procedurii se continuă execuția cu instrucțiunea apelului de procedură; în limbajele de programare, dar și în limbajul pseudocod se utilizează instrucțiunea *return* care provoacă părăsirea corpului unui subprogram;

- *apelarea unei funcții* se face prin considerarea ca *operand* într-o expresie a formei:

  <nume>[<ListăParamActuali>];

  în momentul în care se evaluează expresia respectivă, operandul considerat determină o *corespondență* între parametrii formali și parametrii actuali și se lansează în execuție toate instrucțiunile specificate în cadrul definiției funcției.

Instrucțiuni executabile

Limbajul *pseudocod* oferă următoarele categorii de instrucțiuni executabile:

- *simple*
  - *de intrare / ieșire* (**read**, **write**);
  - *de atribuire / memorare*;
  - *de transfer* (**return**, **exit**);
  - *de operare liste* (*inserare*, *extragere*);
- *structurate*
  - *secvențiale / secvența* (**begin** ... **end**);
  - *decizionale / decizia*:
    - *selecția simplă* (**if** ... **then** ... **else** ...);
    - *selecția multiplă* (**case**);
  - *repetitive / repetiția*:
    - *ciclu cu test inițial* (**while** ... **do** ...);
    - *ciclu cu test final* (**repeat** ... **until** ...);
    - *ciclu cu contor* (**for** ... **do** ...);
  - *apel de procedură / funcție.*

În continuare, fiecare instrucțiune va fi descrisă și prezentată prin *sintaxa* și *semantica* sa.

Instrucțiuni de intrare / ieșire

Sintaxa *instrucțiunilor de intrare / ieșire* este:

read <ListăVariabile>
write <ListăVariabile>

unde <ListăVariabile> este o listă de nume de variabile de tip elementar.

În continuare este descrisă semantica *instrucțiunilor de intrare / ieșire*. Se presupune că citirea valorilor (datelor) se va face de la un mediu de intrare (de exemplu: *tastatură*, *suport magnetic* etc.), iar scrierea rezultatelor (valori de ieșire) se va face la un mediu de ieșire (de exemplu: *display*, *imprimantă*, *plotter*, *suport magnetic* etc.).

Execuția instrucțiunii **read** determină citirea de pe mediul de intrare a valorilor corespunzătoare tipului de variabile ale căror nume sunt indicate în <ListăVariabile>. Ordinea citirii valorilor este dată de ordinea numelor de variabile din <ListăVariabile>.

Execuția instrucțiunii **write** determină scrierea pe mediul de ieșire a valorilor corespunzătoare tipului de variabile ale căror nume sunt indicate în <ListăVariabile>. Ordinea scrierii valorilor este dată de ordinea numelor de variabile din <ListăVariabile>.

Instrucțiunea de atribuire

Pentru instrucțiunea de atribuire / memorare se poate folosi una dintre următoarele două sintaxe:

<var> ← <expresie> sau

(<v_1>, <v_2>, ..., <v_n>) ← (<e_1>, <e_2>, ..., <e_n>),

unde <var>, <v_1>, <v_2>, ..., <v_n> sunt *nume de variabile*, iar <expresie>, <e_1>, <e_2>, ..., <e_n> sunt expresii de același tip (aritmetic, logic, caracter etc.) cu tipul variabilelor corespunzătoare.

Simbolul "←" reprezintă operatorul (operația) de atribuire / memorare. O expresie se construiește folosind operanzi (constante, nume de variabile, apel de funcții), operatori adecvați și eventual paranteze.

Execuția instrucțiunii în prima variantă are ca efect *evaluarea valorii expresiei* din membrul drept, după care în locația corespunzătoare variabilei <var> *se memorează* valoarea obținută în urma evaluării expresiei <expresie>.

În cazul celei de-a doua variante, execuția instrucțiunii este echivalentă cu execuția secvențială / paralelă a instrucțiunilor de atribuire / memorare:

<var_i> ← <e_i>, unde $i \in \{1, 2, \ldots, n\}$.

Instrucțiuni de transfer

Sintaxa instrucțiunii de transfer este:

return [<expresie>] sau
exit

Instrucțiunea **return** poate fi utilizată în cadrul subprogramelor (proceduri și funcții) având rolul în execuție să provoace părăsirea corpului (procesului de calcul) unui subprogram și revenirea în unitatea de program din care a avut loc apelul, și anume la instrucțiunea ce urmează imediat după acest apel, în cazul unei proceduri.

În cazul în care cuvântul **return** este urmat de o expresie, valoarea expresiei obținută prin evaluare este folosită ca valoare de retur a subprogramului.

Instrucțiunea **exit** este utilizată în cazul ieșirii forțate din corpul unei instrucțiuni, și anume, controlul este preluat de instrucțiunea care urmează după instrucțiunea respectivă.

Instrucțiuni de operare cu liste

Sintaxa instrucțiunilor de operare cu liste este următoarea:

<var> ⇒ <lista>
<var> ⇐ <lista>,

unde <var> este un *nume de variabilă* de același tip cu tipul elementelor listei <lista>, iar <lista> este un *nume de listă.*

Prima instrucțiune reprezintă *instrucțiunea de inserare* a elementului <var> în lista <lista>, iar a doua instrucțiune reprezintă *instrucțiunea de extragere* a elementului curent din lista <lista> și memorarea lui în variabila <var>.

Vom prezenta acum sintaxa și semantica unor instrucțiuni care pot fi utilizate în limbajul pseudocod.

Instrucțiuni secvențiale

Sintaxa instrucțiunilor secvențiale este următoarea:

begin
  <instrucțiune$_i$>
  ...
  <instrucțiune$_n$>
end,

unde <instrucțiune$_i$>, i ∈ {1, 2, ..., n} este o instrucțiune executabilă oferită de limbajul pseudocod.

Execuția acestei construcții (secvență de instrucțiuni) înseamnă *execuția secvențială* (*step by step*) a celor *n* instrucțiuni, după care controlul procesului de calcul este dat următoarei instrucțiuni de după cuvântul cheie **end**.

Instrucțiuni decizionale

Există două tipuri de instrucțiuni decizionale, și anume, selecția simplă (**if** ... **then** ... **else** ...) și selecția multiplă (**case**).

Selecția simplă

Sintaxa instrucțiunii de selecție simplă este următoarea:

if <expresie_bool> **then** <corp_1>
  [**else** <corp_2>],

unde <expresie_bool> este o *expresie booleană* (logică), iar <corp_1> și <corp_2> sunt *secvențe de instrucțiuni executabile*.

Inițial se evaluează valoarea logică a expresiei booleene (logică) <expresie_bool>. În cazul în care valoarea obținută este true, atunci (**then**) se trece controlul pentru execuția instrucțiunilor din <corp_1>: altfel (**else**) se trece controlul pentru execuția instrucțiunilor din <corp_2>, în situația când această secvență este prezentă.

În cazul în care valoarea expresiei booleene este false și secvența de instrucțiuni <corp_2> nu apare, controlul este trecut la următoarea instrucțiune de după **if**, acțiune realizată și după execuția instrucțiunilor din <corp_1>, respectiv <corp_2>.

Selecția multiplă

Sintaxa instrucțiunii de selecție multiplă este următoarea:

case <selector> **of**
  C_{11}[, C_{12}, ..., C_{1m}]: <modul$_1$>
  ...
  [C_{n1}[, Cn2, ..., C_{nm}]: <modul$_n$>
  [**else** <modul$_{n+1}$>]
end,

unde <selector> este o *variabilă sau o expresie simplă* de tip diferit de tipul *real*, care va putea avea ca valori constantele distincte C_{ij}, unde i ∈ {1, 2, ..., n}, j ∈ {1, 2, ..., m}, iar <modul$_i$>, i ∈ {1, 2, ..., n, n + 1} este o *secvență de instrucțiuni executabile*.

Prin execuția acestei instrucțiuni, dacă <selector> este o expresie simplă, aceasta va fi evaluată obținându-se o *valoare* care va fi comparată cu constantele C_{ij}, i ∈ {1, 2, ..., n}, j ∈ {1, 2, ..., m}.

În cazul în care <selector> este o variabilă se va considera *valoarea variabilei*. Dacă valoarea este identică unei constante C_{ij}, atunci se trece controlul pentru execuția instrucțiunilor din <modul$_i$> după care se trece la executarea primei instrucțiuni de după **end**.

În cazul în care valoarea nu este identică nici uneia dintre constantele C_{ij}, i ∈ {1, 2, ..., n}, j ∈ {1, 2, ..., m}, atunci se trece controlul pentru execuția instrucțiunilor din <modul$_{n+1}$>, după care se trece la executarea primei instrucțiuni de după **end**.

Instrucțiuni repetitive

Vom prezenta în continuare cele trei tipuri de instrucțiuni repetitive care pot fi utilizate: ciclul cu test inițial (**while** ... **do** ...), ciclul cu test final (**repeat** ... **until**...) și ciclul cu contor (**for** ... **do** ...).

Ciclul cu test inițial

Sintaxa ciclului repetitiv cu test inițial este următoarea:

while <expresie_bool> **do** <corp>,

unde <expresie_bool> este o expresie *booleană* (logică), iar <corp> este o *secvență de instrucțiuni executabile*, inclusiv repetitive.

Execuția acestei instrucțiuni determină evaluarea repetitivă (în cadrul unui proces dinamic) a valorii logice a expresiei <expresie_bool>.

La prima evaluare, dacă valoarea obținută este true, se trece la executarea instrucțiunilor din secvența <corp> și se continuă cu reevaluarea valorii de adevăr (instrucțiunile din secvența <corp> pot schimba starea unor variabile care apar în <expresie_bool>), altfel, dacă valoarea de adevăr este false, nu se mai execută instrucțiunile din secvența <corp>.

În cazul în care instrucțiunile din secvența <corp> s-au executat o dată, se repetă execuția atât timp cât (**while**) valoarea de adevăr a expresiei <expresie_bool> rămâne true.

Procesul repetării executării instrucțiunilor din secvența <corp> se încheie în momentul în care valoarea de adevăr a expresiei <expresie_bool> devine false.

În concluzie, este posibil ca instrucțiunile din secvența <corp> *să nu se execute niciodată* în cazul în care la prima evaluare (*test inițial*) valoarea de adevăr a expresiei <expresie_bool> este false.

De asemenea, se poate face observația că execuția instrucțiunilor din secvența <corp> este *precedată întotdeauna* de testul privind valoarea de adevăr a expresiei <expresie_bool>.

Ciclul cu test final

Sintaxa ciclului repetitiv cu test final este următoarea:

repeat
  <corp>
until <expresie_bool>,

unde <expresie_bool> este o expresie booleană (logică), iar <corp> este o *secvență de instrucțiuni executabile*, inclusiv repetitive.

Execuția acestei instrucțiuni determină execuția repetitivă (în cadrul unui proces dinamic) a instrucțiunilor din secvența <corp> și evaluarea repetitivă a valorii logice a expresiei <expresie_bool>.

La început, necondiționat se trece la executarea instrucțiunilor din secvența <corp> și apoi se continuă cu evaluarea valorii de adevăr (instrucțiunile din secvența <corp> pot schimba starea unor variabile care apar în <expresie_bool>).

Dacă valoarea de adevăr a <expresie_bool> este true, nu se mai execută instrucțiunile din secvența <corp>, altfel se repetă execuția instrucțiunilor din secvența <corp> până când (**until**) valoarea de adevăr a expresiei <expresie_bool> devine true.

Procesul repetării executării instrucțiunilor din secvența <corp> se încheie numai în momentul în care valoarea de adevăr a expresiei <expresie_bool> devine true.

În concluzie, întotdeauna instrucțiunile din secvența <corp> se execută cel puțin o dată și fiecare execuție a instrucțiunilor din secvența <corp> este urmată de evaluarea (*testarea finală*) valorii de adevăr a expresiei <expresie_bool>.

De asemenea, se poate face observația că execuția instrucțiunilor din secvența <corp> este urmată întotdeauna de testul privind valoarea de adevăr a expresiei <expresie_bool>.

Ciclul cu contor

Sintaxa ciclului repetitiv cu contor este următoarea:

for <var> = <exp_inf>, <exp_sup>[, <pas>] **do** <corp>,

unde <var> este o variabilă (numită contorul ciclului) de același tip (excepție tipul *real*) cu tipul expresiilor simple <exp_inf>, <exp_sup> care reprezintă valori (*marginea inferioară*, respectiv *marginea superioară*) dintr-o mulțime de valori ordonate care pot fi generate prin intermediul valorii <pas> (numit *pasul de generare - pasul ciclului*), iar <corp> este o *secvență de instrucțiuni executabile*, inclusiv **for**.

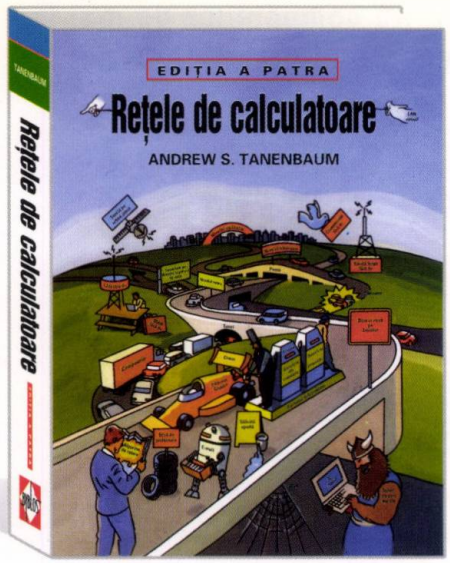
Execuția acestei instrucțiuni determină execuția repetitivă (în cadrul unui proces dinamic) a instrucțiunilor din secvența <corp> prin *modificarea automată* a valorii contorului, având inițial valoarea dată de *marginea inferioară* <exp_inf>, iar apoi valori generate prin acțiunea pasului (prin *incrementare* sau *decrementare*) <pas> până când, după un număr finit de pași, va lua valoarea dată de marginea superioară <exp_sup>.

În momentul când valoarea contorului <var> va fi valoarea dată de marginea superioară <exp_sup>, instrucțiunile din secvența <corp> vor fi executate pentru ultima oară, iar controlul se va transfera la următoarea instrucțiune de după instrucțiunea **for**.

Bibliografie

1. Albeanu Gr., *Algoritmi și limbaje de programare*, Universitatea "Spiru Haret", Editura România de Mâine, București, 2000
2. Apostol C., Roșca I. Gh., Roșca V., Ghilic-Micu B., *Introducere în programare. Teorie și aplicații*, Editura București, 1993
3. Georgescu H., *Programare concurentă. Teorie și aplicații*, Editura Tehnică, București, 1996
4. Kinston J. H., *Algorithms and Structures Design. Correctness, Analysis*, Addison Wesley, Longman, 1998
5. Mitrana V., *Provocarea algoritmilor. Probleme pentru concursurile de informatică*, Editura Agni, București, 1994
6. Odăgescu I., *Optimizarea algoritmilor*, Editura Militară, București, 1991
7. Perjeriu E., Văduva I., *Îndrumar pentru lucrări de laborator la cursul de Bazele Informaticii*, Universitatea din București, 1986
8. Popovici C., Georgescu H., State L., *Bazele Informaticii*, vol. I, Universitatea din București, 1990
9. Skiena S., *The Algorithm Design Manual*, Springer Verlag, New York, Inc., 1998
10. Vlada M., *Informatica*, Universitatea din București, Editura Ars Docendi, București, 1999
11. Vlada M., *Poligoane stelate. Problema lui Hopf și Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348
12. Zaharia M. D., *Structuri de date și algoritmi. Exemple în limbajele C și C++*, Editura Albastră, Cluj-Napoca, 2002
13. Cristea V., Giumale C., Kalisz E., Păunoiu Al., *Limbajul C standard*, Editura Teora, București, 1992
14. Popovici M. D., Popovici M. I., *C++. Tehnologia orientată spre obiecte. Aplicații*, Editura Teora, București, 2000
15. www.math.gatech.edu/~thomas/FC/fourcolor.html
16. mathworld.wolfram.com/Four-ColorProblem.html
17. spicerack.sr.unh.edu/~student/tutorial/fourColor/FourColor.html
18. mathcentral.uregina.ca/RR/database/RR.09.97/fisher1.html
19. www.uccs.edu/~asoifer/book5.html
20. www.uccs.edu/~asoifer/solve98.html
21. www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg
22. mathworld.wolfram.com/Algorithm.html
23. www.cs.rit.edu/~atk/Java/Sorting/sorting.html
24. faculty.juniata.edu/rhodes/cs2/ch129.htm
25. www.scism.sbk.ac.uk/law/Section5/chap1/s5c1p2.htm
26. www.ics.uci.edu/~eppstein/161/people.html
27. www.cs.pitt.edu/~kirk/algorithmcourses/
28. www.gnacademy.org/text/cc/
29. java.sun.com/docs/books/tutorial/java/concepts/
30. www.accu.org/acornsig/public/articles/oop_c.html
31. loki.cs.brown.edu:8081/webae/full.html

Domnul conf. dr. Marin Vlada este cadru didactic la Universitatea București și poate fi contactat prin e-mail la adresa vlada@chem.unibuc.ro.



Cine are carte, are parte!

... de înțelegerea rețelelor!

Cea mai bună introducere din lume în rețele de calculatoare
- complet actualizată pentru tehnologiile cheie de mâine -

Comand-o acum și dacă ești student primești și o reducere semnificativă!

BYBLOS

Byblos srl, tel: 021-330.92.81, fax: 021-330.92.85, e-mail: dpastrama@byblos.ro
www.byblos.ro
Librăria Eminescu, Universitate, la subsol, tel: 021-312.72.65; 0721-927.499

www.agora.ro
www.pcmagazine.ro
www.eweek.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

FORTE COMPUTERS
Flexibility. Made in Romania

IBM®

sat
IT SOLUTIONS & SERVICES

hp®

net consulting

FUJITSU COMPUTERS SIEMENS

SPONSOR DE AUR

CCT
CABINETUL CĂLIN TATOMIR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

gazeta de informatică

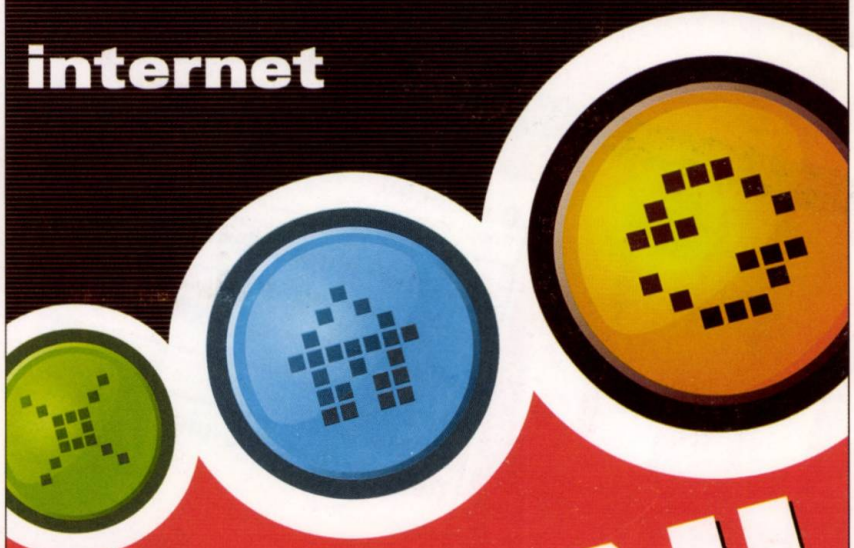
Revista viitorilor specialiști în tehnologia informației

Noiembrie 2003
Vol. 13/7

Lei 30.000

din sumarul numărului:

internet



PAGINI WEB cu PHP 4

Plus

soluții
olimpiada națională de informatică

mate
algoritmi-calculabilitatea funcțiilor

focus
metode de reducere a complexității

Premieră: Parteneriat strategic Microsoft - Guvernul României
PC MAGAZINE
ROMÂNIA
www.pcmagazine.ro
SOLUȚII
Discobolul
Primii pași în Linux
15 sfaturi pentru Excel
Cum să animezi o sigiă
TENDINȚE
Remodelarea PC-ului
Reportaj de la
Intel Developer Forum
Imagine accesibilă
TESTE COMPARATIVE
Plăci video
PC LABS
GHIDUL TĂU ÎN LUMEA TEHNOLOGIILOR
Concurs de soluții!
Microsoft Office
super premii
un PocketPC,
un SmartPhone,
sau un pachet
Office Professional 2003.
20
interfață grafică optimizată
conținut nou substanțial
încărcat 100%
Ghidul tău în lumea tehnologiei
o publicație Agora

Vol. 13, Nr. 7 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Director Bursele Agora:
Mihai Scorțaru
Redactor-șef:
Claudiu Soroiu
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Mihai Stroe

Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
400091, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/592.422, 0265/266.516
Fax: 0265/266.290
http://www.ginfo.ro
ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 0265/262.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
540014 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Vânzare reclamă:
Mihai Bucuroiu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

sumar

soluții.. PAG. 5

Olimpiada NAȚIONALĂ de INFORMATICĂ, ediția 2003

~ soluțiile problemelor de ~
~ la clasele a XI-a și a XII-a ~

Concursul INTERJUDEȚEAN de programare GRIGORE MOISIL

~ soluțiile problemelor ~
~ propuse spre rezolvare ~

probleme.. PAG. 15

ACM

~ enunțurile problemelor ~
~ de la etapa zonală ~

concurs.. PAG. 22

Etapa finală

~ soluțiile problemelor~
~ propuse spre rezolvare ~
~ la etapa finală 2002/2003 ~

mate.. PAG. 24

ALGORITMI - CALCULABILITATE

Horia GEORGESCU

~ calculabilitatea funcțiilor ~
~ prima parte ~

focus.. PAG. 28

Metode de REDUCERE a COMPLEXITĂȚII

Mihai STROE

~ exemple de probleme~
~ pentru care poate fi ~
~ redusă complexitatea~
~ folosind câteva metode~

Algoritmul lui KIRKPATRICK

Mircea DIGULESCU
Andrei MATEI

~ problema localizării ~
~ punctelor în plan ~
~ folosind structura ~
~ lui Kirkpatrick~

internet.. PAG. 34

Pagini WEB cu PHP 4

Mihai SCORȚARU
Claudiu SOROIU

~ operații cu fișiere ~
~ și directoare ~

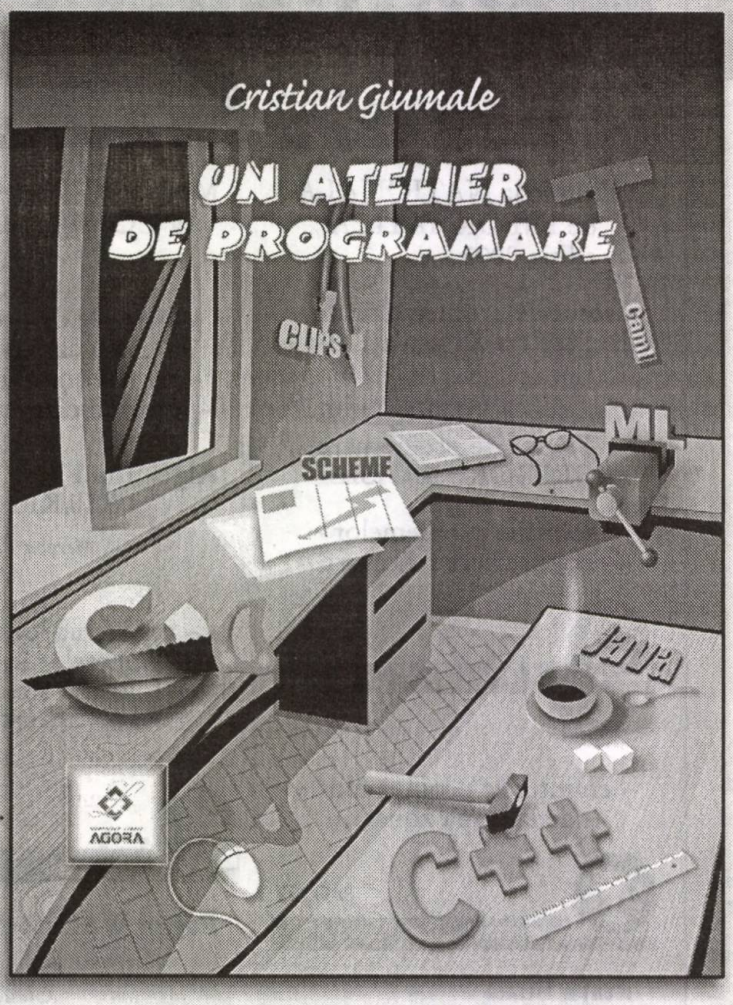
babel.. PAG. 42

Conceptul de ALGORITM - abordare MODERNĂ

Marin VLADA

~ despre algoritmi ~
~ ultima parte ~

COMPUTER LIBRIS AGORA



Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS AGORA

"Mândria" de a fi ELEV

Constată profesorii, dar constată și părinții că elevii sunt din ce în ce mai dezinteresați de școală. (Bineînțeles, vorbim despre fenomen în general. Excepții există întotdeauna.) Acest fenomen se intensifică de la an la an și, pe lângă faptul că elevii lipsesc de la școală, nivelul de pregătire scade.

Apare întrebarea *ce anume cauzează acest fenomen?*

Din păcate, motivul nu este unul singur, iar cauzele se întrepătrund și, astfel, devin complexe. În plus, parcă se formează un cerc vicios. Elevul nu învață, profesorul devine iritat, (constată că muncește aproape inutil) îl ceartă pe elev, acesta, la rândul lui devine și el stresat și nervos, ca atare are motive să *nu* învețe. Dar, atenționarea profesorului ar trebui să-l determine să învețe, nu? De ce să fie ascultător elevul? Cred că răspunsul la această întrebare se poate da și mai greu. Doar de aceea că așa spun adulții? Dar de unde știu ei ce este bine și ce este rău? Din experiența lor de prea multe ori invocată? Este cumva lumea mai bună din cauza că adulții au experiență? Vedeți, răspunsul este complicat...

Majoritatea elevilor consideră că școala este plictisitoare și *nu te învață nimic bun*. Pentru cei care sunt de această părere, nu am decât să le spun că școala nu este plictisitoare, dar ea pare așa pentru că elevul dezinteresat nu știe ce lecții i s-au predat în orele anterioare, deci el nu înțelege nimic la ora la care merge. Dar o oră poate fi plictisitoare și din cauză că profesorul nu reușește să capteze atenția elevilor și aceștia preferă să discute între ei în loc să-l asculte în timp ce acesta reia plictisit o parte din lecție sau și mai rău, țipă la ei, eventual se lamentează că are salar mult prea mic etc.

Elevii care afirmă că *școala nu te învață nimic bun* fac o mare greșeală, deoarece ceea ce ar trebui să se învețe la școală nu se rezumă la lucrurile care se predau la ore, ci există multe altele: Cum explică profesorul *X*? (mai ales dacă nu și-a greșit cariera...) Cum este îmbrăcată profa de franceză? Cum vorbește colegul de bancă despre concertul de aseară? Cum arată afișul pentru balul bobocilor? La școală se petrec lucruri interesante la care participă copii de aceeași vârstă. De ce să te lipsești de această experiență care nu se va mai repeta niciodată în viață?

Greșelile apar tocmai din motivul că, pe de o parte elevii ignoră "viața socială" din școală, pe de alta nu par pasionați de nici un domeniu dintre cele care se studiază acolo. Amintesc încă un motiv pregnant: din păcate la școală nu prea se predă *cum să înveți și cum să gândești singur o rezolvare*. Elevului i-ar plăcea un pic de aventură, un pic de "necunoscut", dar mai ales ar dori să aibă succes. Măcar din când în când...

A învăța nu înseamnă să excelezi în toate domeniile. Acest lucru este practic imposibil. Fiecare elev ar trebui să cunoască foarte bine domeniul care îl pasionează sau pentru care are talent, celelalte domenii ajutându-l să își formeze o cultură generală.

O altă parte a elevilor consideră că sunt lucruri mult mai *bune* de făcut în viață decât mersul la școală. În această situație apare întrebarea: *Ce poate avea un elev mai bun de făcut decât să meargă la școală?* Nu voi afirma că nu există lucruri mult mai bune de făcut, dar școala te învață (direct sau indirect) *cum* să faci acele lucruri.

Pentru cei care nu înțeleg afirmația anterioară îi rog să răspundă la întrebarea: *Ce ați fi făcut acum dacă nu ați fi mers deloc la școală?*.

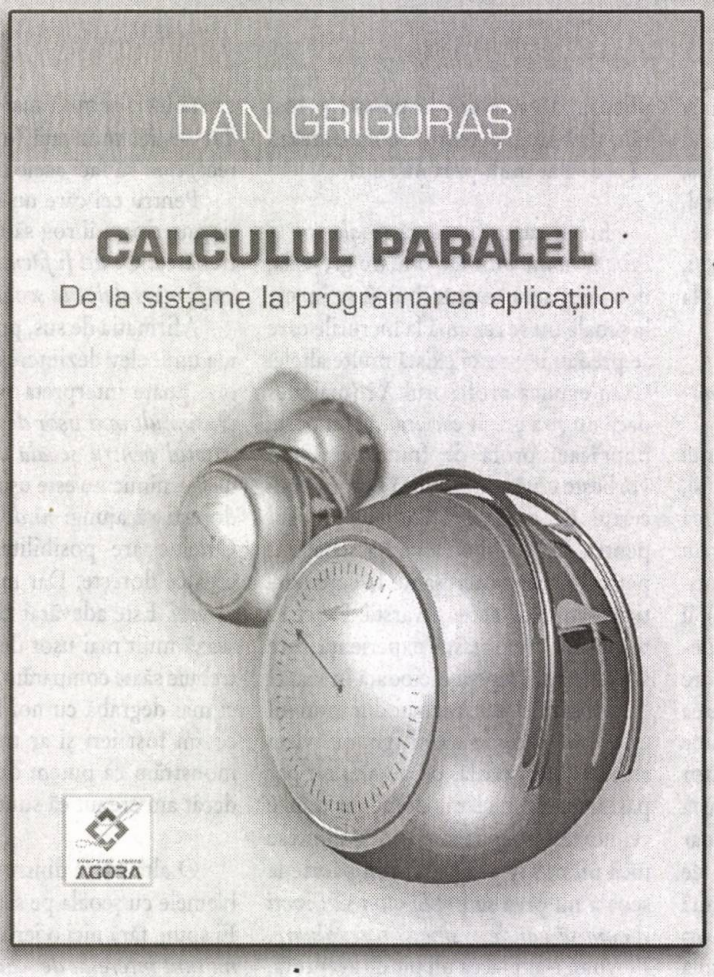
Afirmația de sus, privită prin prisma unui elev dezinteresat de studiu se mai poate interpreta prin "*există lucruri mult mai **ușor** de făcut decât **învățatul** pentru școală*". Da, așa este, dar — nimic nu este ușor în viață, dacă dorești să ajungi *undeva* sau *cineva*. Oricine are posibilitatea să devină ceea ce dorește. Dar mai întîi trebuie să ***vrei***. Este adevărat că unii perseverează mult mai ușor decât noi, dar *nu* trebuie să ne comparăm mereu cu alții, ci mai degrabă cu noi înșine, cu ceea ce am fost ieri și ar trebui să ne demonstrăm că putem deveni mai buni decât am crezut că suntem în stare.

O altă parte dintre elevi lasă problemele cu școala pe seama părinților. Ei spun, fără nici o jenă, ceva de genul *nu sunt interesat de școală, dar nu o să am probleme, deoarece îl cunoaște tata pe cutare și o să vorbească cu el.* Pentru acest gen de elevi problemele vor apărea când vor deveni maturi și în momentul unei angajări nu o să mai fie *tata* cu ei.

Nu am vrut să fac morală nimănui. Voi continua să visez la o lume în care copiii merg la școală cu drag, învață măcar la jumătate din discipline curioși, deoarece știu că ceea ce au "băgat" în creier, acolo rămâne ca în *ROM*.

Claudiu Soroiu

COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS AGORA

Focșani, 18-24 aprilie 2003

Olimpiada NAȚIONALĂ de INFORMATICĂ

Mihai Stroe

Vă prezentăm în continuare soluțiile problemelor propuse la ONI 2003, la clasele a XI-a și a XII-a și la cele două probe de baraj. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de autorii problemelor.

Clasele a XI-a și a XII-a

P050325: A007

Problema se reduce la găsirea unui centru de simetrie într-o mulțime de n puncte având coordonate întregi.

Punctul de simetrie pentru mulțimea de puncte are proprietatea că simetricul fiecărui punct din mulțime față de centrul de simetrie este tot un punct din mulțime. În concluzie, pentru a exista centru de simetrie trebuie ca n să fie par.

Dacă n este par, se observă că punctul căutat (centrul de simetrie) trebuie să fie mijlocul a $n/2$ segmente cu capetele în punctele date. Fiecare punct din mulțime ar fi capăt pentru exact un segment. Dacă am putea identifica aceste segmente, problema s-ar reduce la determinarea mijlocului unuia dintre ele și la testarea dacă acesta este și mijlocul celorlalte.

Notăm cu x și y vectorii care reține coordonatele celor n puncte.

Ordonăm punctele crescător în funcție de abscise. În cazul în care două abscise sunt egale vom ordona crescător în funcție de ordonată.

Pentru un segment cu capetele de coordonate (x_1, y_1) și (x_2, y_2) mijlocul segmentului are coordonatele $x = (x_1 + x_2) / 2$ și $y = (y_1 + y_2) / 2$.

Dacă există centru de simetrie, atunci acesta este plasat în mijlocul segmentului având capetele în primul punct respectiv în ultimul punct (adică cu un capăt în cel mai din stânga punct și celălalt în cel mai din dreapta). Notăm acest punct cu M.

Pentru ca să existe centru de simetrie trebuie ca segmentele având coordonatele (x_i, y_i) și (x_{n-i+1}, y_{n-i+1}) să aibă mijlocul în M, pentru orice $i = 1, 2, ..., n$ div 2. Se verifică aceste condiții pentru punctul M determinat.

Fără a observa legătura dintre sortarea punctelor în ordinea precizată și relația de simetrie, putea fi încercată o abordare bazată pe determinarea repetată a înfășurătorii convexe. Această abordare are complexitatea $O(N^2)$ pe cazul cel mai defavorabil, deci probabil că nu s-ar fi încadrat în timp.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$.

Sortarea punctelor se va realiza cu unul din algoritmii clasici având ordinul de complexitate $O(N \cdot \log N)$.

Operațiile de determinare a mijlocului unuia din segmente și verificarea faptului că acest punct este și mijlocul celorlalte segmente au, în total, ordinul de complexitate $O(N)$.

Operația de scriere a rezultatelor are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N \cdot \log N) + O(N) = O(N \cdot \log N)$.

P050326: Asmin

Pentru această problemă există o soluție a cărei ordin de complexitate este $O(N^2)$. În continuare prezentăm această soluție.

Se consideră fiecare nod (pe rând) ca fiind rădăcină. Pentru rădăcina astfel fixată, valorile asociate nodurilor se pot calcula în timp liniar. Dacă rădăcină este nodul i, atunci valoarea rădăcinii este R_i. Pentru fiecare alt nod, valoarea asociată lui este cea mai mică valoare, astfel încât suma valorilor până la tatăl său în arborele cu rădăcina fixată plus această valoare să dea restul cerut la împărțirea cu numărul K. Valorile sunt atribuite nodurilor de la rădăcină către frunze. Astfel, o simplă parcurgere *DF* (*Depth First* - parcurgere în adâncime) este suficientă, odată ce rădăcina arborelui a fost fixată. Evident, această soluție nu se va încadra în timp pe testele mari.

Soluția optimă are ordinul de complexitate $O(N)$. Se fixează mai întâi ca rădăcină nodul 1 și se calculează valorile asociate nodurilor arborelui având rădăcina în nodul 1, conform algoritmului descris mai sus. Astfel se obține o valoare C_1 egală cu suma valorilor nodurilor arborelui când nodul 1 este rădăcina. Vom numi această valoare *costul asociat nodului* 1.

Pentru a calcula C_i ($i \neq 1$) nu este nevoie să repetăm parcurgerea *DF* pentru nodul i considerat rădăcină. În schimb, încă o parcurgere *DF* din nodul 1 este suficientă pentru a calcula costurile asociate fiecărui nod. Să presupunem că vrem să calculăm C_i și am calculat deja C_j, unde j este tatăl lui i, în arborele având rădăcina fixată în nodul 1.

Atunci, $C_i = C_j - R_j - \text{minim}(R_j, R_i) + R_i + \text{minim}(R_i, R_j)$, unde $\text{minim}(a, b)$ (cu $0 \le a, b \le K\text{-}1$) reprezintă valoarea x minimă ($0 \le x$), astfel încât $a + x = b \pmod K$. Se alege minimul din vectorul C și se tipăresc nodurile i având valoarea C_i egală cu minimul.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$, iar cea de scriere a rezultatului are ordinul de complexitate $O(1)$.

Prima rezolvare constă într-o parcurgere *DF* pentru fiecare nod. Deoarece ordinul de complexitate al unei parcurgeri este $O(N)$, ordinul de complexitate al acestei soluții este $O(N^2)$.

Rezolvarea optimă constă în două parcurgeri *DF*, în care se execută calcule elementare. Ordinul de complexitate al unei astfel de parcurgeri este $O(N)$.

În concluzie, ordinul de complexitate al soluției optime pentru această problemă este $O(N) + O(1) + O(N) = O(N)$.

P050327: Căutare

În prima fază a rezolvării, se citesc și se sortează valorile care vor fi plasate în nodurile arborelui.

Se citesc interogările și, folosind căutări binare, se determină de câte ori se caută fiecare din numerele date și de câte ori se caută numere din intervalul dintre două numere date, consecutive în șirul sortat (sau unul din intervalele exterioare arborelui). Se obține astfel un șir de ponderi.

În continuare, se folosește metoda *programării dinamice*. Se calculează o matrice A, unde $a_{i,j}$ = costul minim pentru a forma un arbore de căutare cu nodurile de la i la j (unde i și j sunt poziții, deci indici pe șirul sortat de valori).

Pentru calculul unei astfel de valori se încearcă fixarea rădăcinii în fiecare poziție k (între i și j inclusiv). Costul arborelui se determină pe baza lui $a_{i,k}$ și $a_{k+1,j}$; determinarea relației exacte este lăsată ca exercițiu pentru cititor.

Valorile $a_{i,j}$ sunt calculate în ordinea crescătoare a diferenței $i - j$; matricea este deci completată pe diagonale. Rezultatul cerut se va afla în $a_{1,N}$.

Ordinul de complexitate al acestei rezolvări este $O(N^3)$ și este dat de cele 3 cicluri `for` imbricate (pentru diferența între i și j, pentru i și pentru k). Timpul de rezolvare pentru un set de date de intrare este rezonabil, dar fișierul de intrare poate conține un număr mare de astfel de seturi de date. În acest caz, rezolvarea nu se va încadra în timp.

Pentru a reduce complexitatea la $O(N^2)$ putem face următoarea observație. Dacă notăm cu $R_{i,j}$ rădăcina arborelui optim care ar conține pozițiile de la i la j din șirul sortat de valori, atunci:

$$R_{i,j-1} \le R_{i,j} \le R_{i+1,j}.$$

Folosind această observație, valorile de pe o anumită diagonală a matricei se pot calcula în timp liniar.

Analiza complexității

În continuare vom analiza complexitatea rezolvării pentru un singur set de date.

Operația de citire a datelor de intrare are ordinul de complexitate $O(N + M)$, iar scrierea rezultatului se realizează în timp constant.

Ordinul de complexitate al primei variante de construire a matricei este $O(N^3)$. Această variantă nu s-ar fi încadrat în timp pentru testele mari.

Folosind relația existentă între valorile din R și calculând elementele de pe fiecare diagonală a matricei în $O(N)$, ordinul de complexitate al construirii matricei este $O(N^2)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru un singur set de date este $O(N + M) + O(1) + O(N^2) = O(N^2 + M)$.

Dacă luăm în considerare toate cele T teste, ordinul de complexitate devine $O(T \cdot (N^2 + M))$.

P050328: Inter

Sunt $N \cdot (N - 1)/2$ puncte de intersecție. Din aceste puncte ne interesează punctele situate în extremități pe fiecare dreaptă. Aceste puncte, cel mult N, sunt vârfurile înfășurătorii convexe a celor $N \cdot (N - 1)/2$ puncte.

Se sortează dreptele după pantă. Se observă că punctele de interes sunt intersecțiile a câte două drepte consecutive în vectorul sortat, și în plus intersecția primei drepte cu ultima.

Se determină înfășurătoarea convexă a acestor N puncte, folosind un algoritm clasic, și se calculează aria zonei rezultate.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(N)$, iar cea de scriere a rezultatului are ordinul de complexitate $O(1)$.

Sortarea dreptelor în funție de pantă are ordinul de complexitate $O(N \cdot \log N)$.

Determinarea punctelor de interes are ordinul de complexitate $O(N)$.

Determinarea înfășurătorii convexe a acestor puncte are ordinul de complexitate $O(N \cdot \log N)$.

Calcularea ariei poligonului obținut are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al rezolvării este $O(N) + O(N \cdot \log N) + O(N) + O(N) + O(N) = O(N \cdot \log N)$.

P050329: Număr

Se determină numărul de apariții pentru fiecare cifră având valori în intervalul [0, 9] în numărul dat x. Acest număr trebuie să fie par (cu excepția eventuală a unei singure cifre; în caz că o astfel de cifră există, trebuie să fie plasată la mijlocul palindromului), altfel nu există soluție.

În continuare se va construi prima jumătate a palindromului; cealaltă jumătate va fi obținută prin oglindirea primei jumătăți.

Se determină numărul de apariții ale fiecărei cifre (0..9) în prima jumătate a numărului x, precum și numărul de apariții necesar (pentru numărul cerut). Dacă pentru o cifră numărul de apariții existent în prima jumătate a numărului dat este prea mare, se elimină aparițiile suplimentare, începând de la dreapta spre stânga.

Se încearcă să se distribuie cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să se obțină cel mai mic număr posibil mai mare sau egal cu x (cu o eventuală rearanjare a cifrelor existente). Distribuirea cifrelor se face căutând de la stânga la dreapta pozițiile cu cifre egale și completarea numărului cu cele din numărul dat.

Când acest lucru nu mai este posibil, ne aflăm în unul din următoarele două cazuri:

- am obținut o jumătate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate). Determinăm anagrama imediat următoare din punct de vedere lexicografic a primei jumătăți a lui x; dacă o astfel de anagramă există, ea (împreună cu cea de-a doua jumătate a palindromului) dă soluția problemei.
- am găsit-o pe cea mai mică cifră mai mare decât cea de pe poziția curentă. Plasăm cifra respectivă, ordonăm crescător cifrele rămase, construim cea de-a doua jumătate și obținem soluția problemei.

Analiza complexității

Ordinul de complexitate al operațiilor de citire a datelor de intrare și furnizare a rezultatelor este $O(N)$.

Determinarea numărului de apariții al fiecărei cifre în numărul x și în numărul dorit au ordinul de complexitate $O(N)$.

Construirea pas cu pas a primei jumătăți a palindromului prin adăugarea unei cifre cât timp este posibil are ordinul de complexitate $O(N)$.

Fiecare din cele două cazuri care încheie rezolvarea se tratează cu algoritmi de complexitate $O(N)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N) + O(N) + O(N) + O(N) = O(N)$.

P050330: Proc

Se observă că timpul minim după care se va termina aplicația este între 0 și $2 \cdot N$ (în cazul când toate fragmentele de cod au timpul 2 și sunt executate pe primul procesor). De asemenea, se observă că, fixând o limită de timp T, se poate determina dacă putem executa toate fragmentele de cod în timpul limită acordat, folosind procesoarele existente. Ne interesează limita minimă de timp pentru care această condiție este respectată.

Se va folosi o căutare binară pentru determinarea limitei de timp. La un moment dat, în cadrul căutării, se va testa dacă putem executa toate fragmentele de cod. Dacă da, atunci se încearcă un T mai mic. Altfel, se mărește T-ul.

Funcția care decide dacă putem executa toate fragmentele de cod în timpul T are complexitatea $O(P)$, unde P este numărul de procesoare disponibile. Se observă că fiecare procesor K asigură $[T/K]$ unități de timp elementare.

Se calculează $N_2 = \sum_{k=1}^{P} [[T/K]/2]$, având semnificația: numărul maxim de fragmente de cod cu timpul 2 care pot fi executate dacă timpul minim este cel mult T. Dacă N_2 este mai mare sau egal cu numărul de fragmente de cod cu timpul 2 existente (să notăm acest număr cu DOI), se verifică dacă se pot executa destule fragmente de cod cu timpul 1. Pentru aceasta se calculează $N_1 = \sum_{k=1}^{P} [[T/K] \bmod 2]$. Dacă $N_1 + 2 \cdot (N_2 - DOI)$ este mai mare sau egal cu numărul de fragmente de cod cu timpul 1, atunci putem executa toate fragmentele de cod în timpul T.

Analiza complexității

Operațiile de citire a datelor și afișarea rezultatelor au ordinul de complexitate $O(1)$.

Sunt $\log N$ pași de căutare binară, deoarece valoarea cautată este limitată superior de $2 \cdot N$.

În cadrul fiecărui pas se execută o funcție care are ordinul de complexitate $O(P)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(1)+O(\log N) \cdot O(P) = O(P \cdot \log N)$.

Baraj

P050331: Excursie

Această problemă admite o rezolvare matematică. Se evidențiază următoarele cazuri pe care le prezentăm în continuare.

În cazul în care $K > P$, atunci rezultatul este 0.

Dacă $P = K$, vom distribui cei K ghizi în primele K excursii. Rămân $N - K$ persoane, care pot fi repartizate în cele K excursii disponibile. În concluzie, există K^{N-K} posibilități.

Ultimul caz corespunde relației $P > K$.

Să notăm $r = P - K$, $r \geq 1$.

Repartizăm mai întâi cei K ghizi în K excursii distincte (acestea vor fi excursiile 1, 2,..., K).

Rămân $N - K$ persoane. Dintre acestea, unele vor fi repartizate în cele $P - K = r$ excursii care nu au ghizi și celelalte vor fi repartizate în excursii care au ghizi.

Notăm cu x numărul de persoane care vor fi repartizate în excursii care nu au ghizi. Acestea pot fi partiționate în r excursii în $S(x, r)$ moduri (cu $S(x, r)$ notăm *numerele lui Stirling de speța a II-a*).

Deci există $N - K - x$ persoane care trebuie repartizate în cele K excursii care au ghizi. Acest lucru se poate realiza în K^{N-K-x} moduri.

Deoarece putem selecta x persoane din cele $N - K$ rămase în C_{N-K}^{x} moduri, rezultă că se obțin:

$NR = \sum_{x=r}^{N-K} \left(C_{N-K}^{x} \cdot S(x,r) \cdot K^{N-K-x} \right)$ posibilități.

Numerele lui Stirling de speța a II-a, $S(n, m)$ reprezintă numărul de partiții ale unei mulțimi cu n elemente în m clase ($n \le m$). Evident, clasele partiției trebuie să fie nevide.

Pentru a determina numerele lui *Stirling de speța a II-a* se poate utiliza următoarea relație de recurență:

$S(n, 1) = S(n, n) = 1$;

$S(n, m) = S(n - 1, m - 1) + m \cdot S(n - 1, m)$.

Remarcăm că pentru rezolvarea problemei nu este necesară cunoașterea în prealabil a numerelor lui *Stirling* .

Analiza complexității

Operațiile de citire și scriere a datelor au ordinul de complexitate $O(1)$.

Cazul cel mai defavorabil, din punct de vedere al complexității se obține pentru $P > K$.

Pentru calculul rezultatului se folosesc numere mari și vom considera că folosim numere mari de M cifre.

Calculul numerelor *Stirling* de speța a II-a are ordinul de complexitate $O(N \cdot P \cdot M)$. Se folosesc operațiile de adunare a numerelor mari și înmulțire a unui întreg mic cu un număr mare.

Combinările din cadrul formulei se pot calcula într-un timp $O(N^2 \cdot M)$, folosind formula $C(n, k) = C(n - 1, k) + C(n - 1, k - 1)$ și operația de adunare a numerelor mari.

Puterile lui K se pot calcula cu un algoritm de complexitate $O(N \cdot M)$.

Rezultatul final se poate calcula cu un algoritm de complexitate $O(N \cdot M^2)$, folosind operația de înmulțire a numerelor mari.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N \cdot M^2)$. Nu am ținut cont de modul în care crește numărul M de cifre pe parcursul rezolvării, dar îl putem considera ca fiind tot timpul egal cu cea mai mare valoare pe care o poate atinge.

P050332: Robot

Pentru a rezolva această problemă pornim de la următoarele observații.

Regiunea accesibilă unui braț format dintr-un singur segment de lungime L_1 este, în mod evident un cerc cu raza L_1 și centrul în origine (umărul robotului).

Regiunea accesibilă unui braț format din două segmente de lungimi L_1, respectiv L_2 este un inel, cu raza exterioară $L_1 + L_2$ și raza interioară $|L_1 - L_2|$.

Regiunea de accesibilitate a unui braț format din n segmente este un disc centrat în origine care are raza exterioară $ro = L_1 + L_2 + \ldots + L_n$ (corespunzător alinierii tuturor segmentelor). Raza interioară este 0 dacă lungimea celui mai lung segment L_{max} este mai mică decât jumătate din lungimea totală a brațului. În caz contrar, raza interioară este egală cu $2 \cdot L_{max} - ro$. Se poate demonstra că orice punct din acest disc este accesibil.

Cu aceste observații, în continuare trebuie construită o configurație a brațului care să permită atingerea punctului dorit.

Dacă brațul are două segmente, de lungimi L_1 și L_2, problema se reduce la o simplă intersecție de cercuri. Fie p punctul care trebuie să fie atins. Intersectăm cercul C_1 de rază L_1, centrat în origine, cu cercul C_2, de rază L_2, centrat în p. În general pot exista 0, 1, 2 sau o infinitate de puncte de intersecție în cazul în care cele două cercuri coincid. Fie R un punct de intersecție. Pentru a determina configurația brațului, trebuie să calculăm unghiul dintre axele OX și OR.

Dacă brațul are trei sau mai multe segmente, se poate demonstra că orice punct din mulțimea punctelor accesibile poate fi atins dacă îndoim brațul în cel mult două locuri, și anume în extremitățile segmentului care conține mijlocul brațului, dacă brațul ar fi întins.

Prima impresie ar fi că o singură îndoire este suficientă, dar există contraexemple clare care demonstrează incorectitudinea acestui raționament.

După ce am stabilit unde îndoim brațul, având la dispoziție centrul și punctul care trebuie atins, configurația brațelor se determină cu ajutorul unor calcule simple.

Analiza complexității

Operațiile de citire și scriere a datelor au ordinul de complexitate $O(N)$.

Complexitatea algoritmului este $O(N)$, datorită calculării sumei lungimilor segmentelor și a poziției mijlocului brațului. Calculele matematice care urmează se realizează în timp constant.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N) + O(N) = O(N)$.

P050333: Telegraf

Personal, problema mi s-a părut foarte dificilă și, cel puțin în concurs, nu aș fi știut să o rezolv. În aceste condiții, ea putea fi tratată ca un exercițiu de "*furat puncte*", cu rezultate mai mult sau mai puțin satisfăcătoare. Oricum, nivelul de dificultate al problemelor crește de la an la an, fapt ce se evidențiază și în rezultatele excelente obținute de elevii români participanți la concursurile internaționale.

Soluția problemei, descrisă mai jos, este construită numai pe baza indicațiilor autorului problemei.

Pentru cunoscători, problema seamănă cu codificarea *Huffman*, diferența (esențială) constând în faptul că unul din simbolurile elementare din textul codificat necesită două unități de timp pentru a fi transmis.

Soluția problemei constă în determinarea unui arbore binar, în care:

- frunzele sunt asociate simbolurilor din șir;
- deplasarea dintr-un nod spre stânga corespunde unui punct, iar spre dreapta, unei linii;
- durata transmisiei unui simbol este dată de lungimea drumului de la rădăcină la frunza asociată. Punctul contribuie cu 1 la lungime, iar linia, cu 2.

Simbolurile fiind asociate frunzelor, textul transmis codificat nu este ambiguu (reprezentarea unui simbol din șirul inițial nu poate fi prefixul reprezentării unui alt simbol).

În plus, se observă că arborele este binar strict. Asftel, dacă ar exista în arbore un nod cu un singur descendent, putem șterge acest nod și îl înlocuim cu arborele care are ca rădăcină descendentul, obținând o codificare cu timp de transmisie mai mic.

Dacă vom considera rădăcina pe nivelul 0, fiul stâng al unui nod de pe nivelul i fiind situat pe nivelul $i + 1$, iar fiul drept pe nivelul $i + 2$ (datorită lungimii diferite a muchiilor), se observă că o parcurgere pe niveluri a arborelui va lista simbolurile în ordinea descrescătoare a frecvenței. Altfel spus, simbolurile care apar mai des vor fi situate mai aproape de rădăcină.

Folosind aceste observații putem construi un algoritm bazat pe *metoda programării dinamice*. Vom construi (conceptual) arborele pe niveluri de sus în jos. Următoarele informații reprezintă starea:

- A - numărul de noduri interne care sunt pe nivelul $L - 1$;
- B - numărul de noduri interne care sunt pe nivelul $L - 2$;
- F - numărul de frunze care au apărut deja pe nivelurile mai mici decât L.

Este posibil să găsim optimul pentru o anumită configurație (A, B, F), la orice nivel s-ar afla. Bineînțeles, dacă nu știm valoarea lui L nu putem calcula efectiv costul unei frunze când o construim. De aceea costurile trebuie adunate "pe parcurs": când coborâm cu un nivel în arbore, vom aduna costul unui nivel în contul tuturor frunzelor aflate mai jos decât nivelul respectiv. Evident, frunzele situate mai jos de nivelul L au indicii $F + 1$, $F + 2$, ... (dacă am ordonat simbolurile descrescător în funcție de frecvența lor de apariție).

Deci, costul unui nivel L va fi dat de suma frecvențelor simbolurilor începând cu $F + 1$; aceste sume le vom calcula la începutul algoritmului.

Așa cum am menționat, algoritmul construiește un astfel de tablou tridimensional folosind *metoda programării dinamice*. Memoria folosită este $O(N^3)$, unde am folosit N pentru numărul de simboluri (36 în cazul de față). Cel mai simplu mod de a calcula elementele tabloului este o soluție recursivă cu memoizare. Pentru aceasta, trebuie să vedem din ce poate proveni o soluție (A, B, F). Singura variabilă este numărul de frunze de pe nivelul L; pentru X frunze pe nivelul L, starea precedentă este $(B - A, A, F - X)$. Aceasta ne dă un algoritm în timp $O(N^4)$ pentru construirea tabloului.

Analiza complexității

Notăm cu N numărul de simboluri cu frecvențe strict pozitive. În enunțul acestei probleme, N este limitat superior de 36.

Datorită limitei superioare pentru numărul N, citirea datelor are ordinul de complexitate $O(36)$.

Operația de construire a matricei, prin metoda programării dinamice, are ordinul de complexitate $O(N^4)$.

Operațiile de găsire a optimului se pot realiza în timpul construirii matricei.

Operația de afișare a rezultatelor are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(36) + O(N^4) + O(1) = O(N^4)$.

P050334: Ajutor

Problema se poate rezolva în mai multe moduri. În continuare vom prezenta două soluții ale acesteia.

Se consideră mai întâi cele N puncte de prim ajutor. Se iau coordonatele x ale acestor puncte într-un vector, care este sortat. Se face același lucru și cu coordonatele y. Fiecare punct având abscisa în vectorul x și ordonata în vectorul y va reprezenta un nod într-o rețea ortogonală (caroiaj). Pentru aceste noduri se calculează, folosind *metoda programării dinamice*, distanța până la cel mai apropiat post de prim ajutor.

Se folosește metoda programarii dinamice de patru ori, începând din stânga-sus, apoi începând din stânga-jos, apoi începând din dreapta-jos, apoi începând din dreapta-sus. Calculele se fac (conceptual) cu ajutorul unor matrice $a[i, j]$, reprezentând distanța până la cel mai apropiat post de prim ajutor, dacă am merge, de exemplu, numai în stânga și în sus (pentru prima matrice), din pozitia (i, j) din caroiaj, adică a i-a coordonată y, după sortare și a j-a coordonată x după sortare. Calculul se face $O(N^2)$, pentru că, de exemplu, valoarea lui $a[i, j]$ nu depinde decât de vecinii din stânga-sus.

Dintre cele patru valori se reține minimul și se memorează valoarea sa într-o matrice.

Pentru a calcula valorile respective nu este nevoie de matrice, ci de doi vectori: nivelul curent și nivelul anterior.

Fiecare dintre cele M locații ale accidentului se rezolva (se află minimul) în $O(\log N)$, pentru că nu trebuie decât să aflăm în care celulă din caroiaj se află punctul în care ne situăm (două căutări binare în vectorii x și y), iar acolo să vedem în care dintre cele 4 colțuri este mai bine să mergem.

Complexitatea acestei variante de rezolvare este, în consecință, $O(N^2 + M \cdot \log N)$

În cele ce urmează vom prezenta o altă soluție care are ordinul de complexitate mai mic.

Datorită limitărilor coordonatelor pentru puncte, apare și posibilitatea de a răspunde la o cerere în $O(1)$. Acest lucru se face prin preprocesarea vectorilor x și y. Se mai reține două tablouri în care, pentru fiecare coordonată x, se află coloana i și, pentru fiecare coordonată y, se află linia j a caroiajului obținut prin preprocesare.

Este nevoie deci de două tablouri a câte 32000 de elemente de tip `word` (respectiv, `unsigned int`). În acest caz, răspunsul pentru fiecare locație posibilă a accidentului poate fi dat în timp constant (după rezolvarea subproblemei de programare dinamică).

Complexitatea acestei variante de rezolvare a problemei este $O(N^2 + M)$.

Analiza complexității

Operația de citirea a datelor are ordinul de complexitate $O(M + N)$, iar cea de scriere a rezultatelor are ordinul de complexitate $O(M)$.

Preprocesarea, constând în aplicarea *metodei programării dinamice* pentru a calcula distanțele minime în cazul caroiajului, are ordinul de complexitate $O(N^2)$.

Preprocesarea pe axele x și y, din cadrul celei de-a doua variante de rezolvare, are ordinul de complexitate constant (datorită limitării la 32000).

Pentru fiecare locație posibilă a accidentului, răspunsul se dă în timp constant în varianta 2, în timp ce în cazul primei variante de rezolvare ordinul de complexitate este $O(\log N)$ datorită căutării binare.

În concluzie, ordinul de complexitate al rezolvării este $O(M + N) + O(N^2) + O(M) \cdot O(\log N) + O(M) = \mathbf{O(N^2 + M \cdot \log N)}$ în cazul primei variante, respectiv $O(M + N) + O(N^2) + O(M) + O(M) = \mathbf{O(N^2 + M)}$ pentru cea de-a doua variantă.

P050335: Ghizi

Se observă că problema devine o problemă de flux într-o rețea de transport dacă facem următoarele transformări:

- momentele întregi de timp se transformă în nodurile unui graf;
- un interval $[T_1, T_2)$ asociat unui ghid introduce un arc de la T_1 la T_2, cu capacitatea 1. Două noduri pot fi conectate prin mai multe arce.

Se observă că un drum de la sursă (nodul 0) la destinație (nodul 100) acoperă fiecare moment de timp din [0, 100) exact o dată. Astfel, problema poate fi reformulată:

"Dându-se graful construit conform transformării de mai sus, să se determine K drumuri, disjuncte din punct de vedere al muchiilor, de la sursă la destinație."

Problema este în mod evident o problemă de flux; se introduce un arc de capacitate K de la o sursă fictivă la nodul 0 și se execută un algoritm de flux maxim.

Trebuie aleasă o variantă de implementare care să se încadreze în timpul de execuție acordat. Nu descriem algoritmii de flux deoarece, pentru scopul acestui articol, îi considerăm cunoscuți.

Analiza complexității

Notăm cu T numărul de momente distincte de timp care pot fi capete ale unui interval asociat unui ghid (maxim 101).

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(N)$.

Construirea grafului are, de asemenea, ordinul de complexitate $O(N)$. Se obține un graf cu T noduri și N muchii.

Aplicarea algoritmului de flux maxim *Ford-Fulkerson* presupune găsirea a K drumuri de creștere de la sursă la destinație. Presupunând că fiecare drum se determină printr-o singură parcurgere a grafului cu complexitatea $O(N)$, găsirea celor K drumuri de creștere are ordinul de complexitate $O(N \cdot K)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N) + O(N \cdot K) + O(N) = \mathbf{O(N \cdot K)}$.

P050336: Sala

Se poate demonstra, prin inducție matematică (cu pasul 3), că numărul maxim de băieți în sală este egal cu:

$$x_n = [(n \cdot (n+1)+1)/3].$$

Pentru $n \le 3$ verificările sunt imediate.

Să considerăm afirmația adevărată pentru n și să o demonstrăm pentru $n + 3$.

Fie $a_1, a_2, \ldots, a_n, a_{n+1}, a_{n+2}, a_{n+3}$ o așezare pentru $n + 3$ persoane. Următoarele două rânduri sunt date prin vectorii:

$b_1, b_2, \ldots, b_n, b_{n+1}, b_{n+2}$

$c_1, c_2, \ldots, c_n, c_{n+1}$

Pentru aceste trei rânduri de persoane numărul minim de fete este $n + 2$, deci numărul maxim de băieți este $2 \cdot n + 4$.

Astfel obținem $x_{n+3} \le 2 \cdot n + 4 + x_n$.

Conform ipotezei de inducție se obține că:

$$x_{n+3} \le 2 \cdot n + 4 + [(n \cdot (n+1)+1)/3] = [((n+3) \cdot (n+4)+1)/3].$$

Relația de mai sus reprezintă exact ceea ce trebuia demonstrat.

Valoarea maximă $x_n = [(n \cdot (n+1)+1)/3]$ se poate obține pentru următoarea așezare în sală:

```
bbfbbfbbf...
fbbfbbfb...
bfbbfbb...
...
b
```

Calculul valorii necesită implementarea operațiilor cu numere mari, însă programul poate fi redactat ușor.

Analiza complexității

Notăm cu K numărul de cifre al numărului dat la intrare.

Operația de citire a datelor de intrare are ordinul de complexitate $O(K)$.

Pentru calculul formulei folosim înmulțirea pe numere mari, de complexitate $O(K^2)$.

Operația de afișare a rezultatului are ordinul de complexitatea $O(K)$, deoarece rezultatul nu poate avea mai mult de $2 \cdot K$ cifre.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(K) + O(K^2) + O(K) = \mathbf{O(K^2)}$.

Observații

Toate variabilele, care au fost folosite în rezolvarea acestor probleme și nu au fost explicate, au semnificația dată în enunțurile problemelor, care au fost publicate în *GInfo* 13/5 (mai 2003).

Satu Mare, 29 martie 2003

Concursul INTERJUDEȚEAN de programare GRIGORE MOISIL

Silvestru-Cosmin Negruşeri

Vă prezentăm în continuare soluțiile problemelor propuse la concursul interjudețean de programare Grigore Moisil. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de autorii problemelor.

Clasele a IX-a

P060319: Excursie

Pentru această problemă se poate observa foarte ușor că numărul de posibilități în care se poate urca până în vârful muntelui este egal cu F_{n-3}, unde n este numărul de serpentine și F_i, i = 1, 2, 3, ..., n este șirul lui *Fibonacci* care respectă următoarele relații:

- $F_0 = 0$;
- $F_1 = 1$;
- $F_i = F_{i-1} + F_{i-2}$, $i > 1$.

Deci, dacă avem o singură serpentină, există trei trasee distincte care respectă condițiile impuse în enunțul problemei. Dacă avem două serpentine, există cinci trasee.

Dacă nu ar fi fost impusă restricția $n \leq 40$ asupra numărului de serpentine n, ar fi apărut necesitatea implementării operației de adunare cu numere mari.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor are ordinul de complexitate $O(1)$.

Calcularea celui de-al $(n - 3)$-lea element din șirul lui *Fibonacci* folosind relația de recurență are ordinul de complexitate $O(n)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1) + O(n) + O(1) = O(n)$.

P060320: Loto

Prima cerință a problemei este de a determina numărul total de variante pe care le poate juca *Vasile* în noul joc creat de el. Acest număr reprezintă numărul de submulțimi de y elemente ale unei mulțimi cu x elemente și este egal cu C_x^y (combinări de x elemente luate câte y).

Există mai multe metode de determinare a acestui număr. Datorită faptului că datele de test sunt mici chiar și o rezolvare care generează toate variantele posibile s-ar încadra în timp.

Soluția oficială a problemei folosește șase structuri repetitive `for` imbricate datorită faptului că numărul de numere prezente într-o variantă nu depășește valoarea 6 și generează toate variantele posibile. Un astfel de algoritm are ordinul de complexitate $O(C_x^y)$.

Această cerință se mai poate rezolva folosind formula de calcul pentru combinări care este $C_x^y = x!/(y! \cdot (x - y)!)$, dar la început va trebui să simplificăm cu cea mai mare valoare de la numitor și apoi, la fiecare pas trebuie efectuate împărțiri, sau folosind următoarea relație de recurență:

- $C_n^0 = 1$;
- $C_n^n = 1$;
- $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$.

Cea de-a doua cerință a problemei este să determinăm numărul variantelor care rămân nejucate dintre cele n posibile, după ce am extras variantele care le vom juca pe baza regulii impuse în enunț. Considerăm toate variantele posibile ca făcând parte dintr-o listă circulară și jucăm variantele din k în k până când ajungem la o variantă pe care am jucat-o deja.

Deci, pentru a rezolva această cerință, trebuie să determinăm valoarea minimă x având proprietatea că $x \cdot k$ este multiplu al lui n. Valoarea minimă pentru x, care respectă proprietate cerută, este $cmmmc(n, k) / k$, unde $cmmmc(n, k)$ este cel mai mic multiplu comun al numerelor n și k.

Valoarea x astfel determinată reprezintă chiar numărul de variante jucate, deci numărul de variante care nu s-au jucat încă este egal cu $n - x$.

Pentru a putea calcula *cmmmc*(n, k) se folosește relația *cmmmc*$(n, k) = m \cdot n$ / *cmmdc*(n, k), unde *cmmdc*(n, k) este cel mai mare divizor comun al numerelor n și k și poate fi calculat folosind algoritmul lui *Euclid*.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(1)$.

Ordinul de complexitate pentru calculul combinărilor pentru prima cerință a problemei, dacă se folosește relația de recurență, este $O(x \cdot y)$.

Ordinul de complexitate al algoritmului de rezolvare pentru cea de-a doua cerință este dat de ordinul de complexitate al algoritmului de calcul pentru *cmmdc*(n, k) care este $O(\log (n + k))$, restul operațiilor se realizează în timp constant.

În concluzie, ordinul de complexitate a soluției pentru această problemă este $O(1) + O(x \cdot y) + O(\log (n + k)) + O(1) = O(x \cdot y) + O(\log (n + k))$.

Clasele a X-a

P060321: Împărțeală

Problema se reduce la determinarea numărului de partiții a numărului n în k numere care să fie ordonate crescător.

Rezolvarea problemei se bazează pe *metoda programării dinamice*. Un algoritm de rezolvare care se bazează pe metoda *backtracking* nu s-ar încadra în timp pentru toate testele.

Notăm cu $num_{i,j}$ numărul de partiții a numărului i în j numere. Observăm că $num_{i,j}$ respectă următoarea relație de recurență:

$$num_{i,j} = \sum_{p=0}^{\lfloor i/j \rfloor} num_{i-j \cdot p, j-1}.$$

În continuare vom explica de unde a apărut această relație. Observăm că numărul de soluții care au primul termen egal cu p este egal cu numărul de partiții crescătoare de k termeni care au fiecare termen mai mare sau egal cu p. Dacă scădem din fiecare element al partiției numărul p obținem numărul de partiții crescătoare cu primul termen 0 al numărului $n - k \cdot p$.

În momentul implementării soluției algoritmului trebuie ținut cont de faptul că elementele aflate pe prima coloană a matricei *num* respectă următoarea relație: $num_{i,1} = 1$, $1 \le i \le n$, deoarece numărul de partiții cu un element pentru numărul i este egal cu 1. Rezultatul pentru această problemă îl va constitui valoarea $num_{n-k,k}$ deoarece fiecare termen al partițiilor trebuie să fie mai mare sau egal cu 1.

Analiza complexității

Operațiile de citire a datelor de intrare, respectiv de scriere a rezultatelelor au ordinul de complexitate $O(1)$.

Determinarea numărului de partiții are ordinul de complexitate $O(k \cdot n \cdot \log n)$, deoarece calculul tuturor sumelor la pasul curent are ordinul de complexitate $O(n \cdot \log n)$, ordinul mediu de complexitate al operației de determinare a unei singure sume fiind $O(\log n)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1) + O(k \cdot n \cdot \log n) + O(1) = O(k \cdot n \cdot \log n)$.

P060322: Cutii

Această problemă a fost propusă în altă formă la o selecție a lotului olimpic din 1999 și o rezolvare a fost publicată în paginile revistei *GInfo*.

Pentru a rezolva problema putem folosi *metoda greedy*, astfel: punem cutia curentă în cea mai mică cutie, care este mai mare decât ea, existentă printre cutiile deja poziționate.

Șirul dimensiunilor cutiilor deschise este crescător. Această afirmație poate fi demonstrată ușor prin inducție și prezentăm în continuare această demonstrație.

Pentru un singur pachet afirmația este adevarată. În cazul în care avem un șir de n pachete ordonate crescător și adăugăm o cutie nouă, atunci ea fie formează un nou pachet (deci șirul este crescător), fie intră într-un pachet deja format, și din modul de alegere a pachetului rezultă că șirul dimensiunilor rămâne crescător.

În concluzie, dacă folosim proprietatea că șirul este crescător, pentru a găsi locul cutiei curente vom folosi o căutare binară în locul unei căutări secvențiale.

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(n)$.

Operația de împachetare a cutiilor are ordinul de complexitate $O(n \cdot \log n)$, deoarece pentru fiecare cutie care vine pe bandă căutăm locul unde va fi poziționată folosind o căutare binară.

În concluzie, ordinul de complexitate a soluției acestei probleme este $O(n) + O(n \cdot \log n) + O(n) = O(n \cdot \log n)$.

Clasele a XI-a și a XII-a

P060323: Ali Baba și cei 40 de hoți

Acestă problemă se reduce la determinarea elementului majoritar dintr-un șir. Restricțiile impuse în enunțul problemei ne permit să sortăm șirurile și să determinăm elementul majoritar. Pentru restricții mai puțin severe, o metodă de rezolvare a fost tratată într-un articol mai vechi din *GInfo*. O altă rezolvare a fost dată de studenții de la *Universitatea București* în cadrul concursului regional *ACM* din anul 1998, și această rezolvare fiind publicată în paginile *GInfo*.

Pentru o a treia rezolvare observăm că, dacă șirul mărgelelor ar fi sortat, atunci culoarea mărgelei majoritare (dacă există o mărgea majoritară) ar fi chiar la mijlocul șirului. Însă pentru determinarea elementului median dintr-un șir nu este necesar să sortăm șirul ci putem să folosim abordarea randomizată care are ordinul de complexitate $O(n)$.

Subalgoritmul de determinare a numărului median dintr-un șir care are ordinul de complexitate $O(n)$ este similar subalgoritmului de sortare rapidă în care pivotul se alege aleator și este folosit pentru a determina al i-lea cel mai mic element. Acest algoritm este:

```
Subalgoritm DeterminăMediana(a,p,r,i)
// a - șirul de numere
// p, r - indicele elementelor între care se caută al i-lea cel
          mai mic element
dacă p = r atunci
   returnează a_p
q ← PartițieAleatoare(a, p, r)
k ← q - p + 1
dacă i ≤ k atunci
   returnează DeterminăMediana(a,p,q,i)
altfel
   returnează DeterminăMediana(a,q,r,i - k)
sfârșit subalgoritm
```

Acest subalgoritm se apelează cu parametrii a, 1, n, respectiv $[n / 2]$, pentru un șir de numere a cu n elemente.

Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate $O(n \cdot nr)$, unde n este numărul de boluri, iar nr este numărul mediu de mărgele care se află în boluri.

Operația de determinare a elementului majoritar are ordinul de complexitate $O(n \cdot nr)$.

Operația de scriere a rezultatelor de intrare are ordinul de complexitate $O(k \cdot n)$, unde k este numărul maxim de mărgele identice care se extrag din fiecare bol.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(n \cdot nr) + O(n \cdot nr) + O(k \cdot n) = O(n \cdot (k + nr))$.

P060324: Profi și studenți

Structura profesorilor formează un arbore, iar procedeul, prin care aceștia părăsesc sala de ședințe, explicat în enunțul problemei, reprezintă algoritmul de determinare al codului *Prufer* pentru un arbore.

Se observă că o rezolvare cu ordinul de complexitate $O(n^2)$ s-ar încadra în timpul acordat pentru execuție.

În continuare vom încerca o abordare al cărei ordin de complexitate este $O(n \cdot \log n)$ pentru a determina ordinea de ieșire a profesorilor.

Vom folosi un șir g cu n elemente, unde n reprezintă numărul de noduri din arbore care este egal cu numărul de profesori, iar g_i, $1 \le i \le n$, reprezintă numărul de fii ai nodului i. Vom mai folosi o structură *min-heap* (*heap* în care cel mai mic element este vârful) în care vom introduce nodurile pentru care $g_i = 0$, care reprezintă frunzele arborelui.

În continuare, analizăm codul *Prufer* secvențial și, la fiecare pas, eliminăm frunza care are asociat numărul cel mai mic, decrementăm gradul nodului tată, iar dacă gradul a devenit 0, îl inserăm în *min-heap*-ul existent.

Pentru decodificarea codului *Prufer* în $O(n)$ se poate studia ***Problema 8*** din cartea *Psihologia concursurilor de informatică*, Cătălin Frâncu, Editura L&S, București, 1997.

Analiza complexității

În continuare vom analiza complexitatea algoritmului de rezolvare pentru o singură universitate.

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(n)$.

Complexitatea algoritmului de decodificare a codului *Prufer* prezentată este $O(n \cdot \log n)$.

În concluzie, ordinul de complexitate al algoritmului de decodificare a codului *Prufer* este $O(n) + O(n \cdot \log n) + O(n) = O(n \cdot \log n)$.

Dacă numărul de universități este m, atunci ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(m \cdot n \cdot \log n)$.

P060325: Hacker

Această problema reprezintă o aplicație directă a *metodei programării dinamice*.

Vom nota cu s șirul care reprezintă cheia de criptare, cu a matricea citită din fișierul de intrare și cu n dimensiunea acesteia.

Vom folosi un tablou auxiliar bidimensional c care are aceleași dimensiuni cu cele ale matricei citite, iar c_{ij} reprezintă numărul de apariții al subșirului lui s care începe de pe prima poziție din șir și se termină pe poziția pe care se află elementul a_{ij} în s. La început, elementele matricei pentru care elementele corespunzătoare din matricea a sunt egale cu primul caracter din cheia s vor fi inițializate cu valoarea 1, iar restul cu valoarea 0.

În continuare, vom parcurge secvențial cheia s, iar elementele din c, pentru care elementul corespunzător din matricea a este egal cu caracterul curent din cheia s, vor primi ca valoare suma vecinilor lor.

În final, rezultatul problemei îl constituie suma tuturor elementelor din matricea c al căror corespondent din matricea a este egal cu ultimul caracter din s.

Analiza complexității

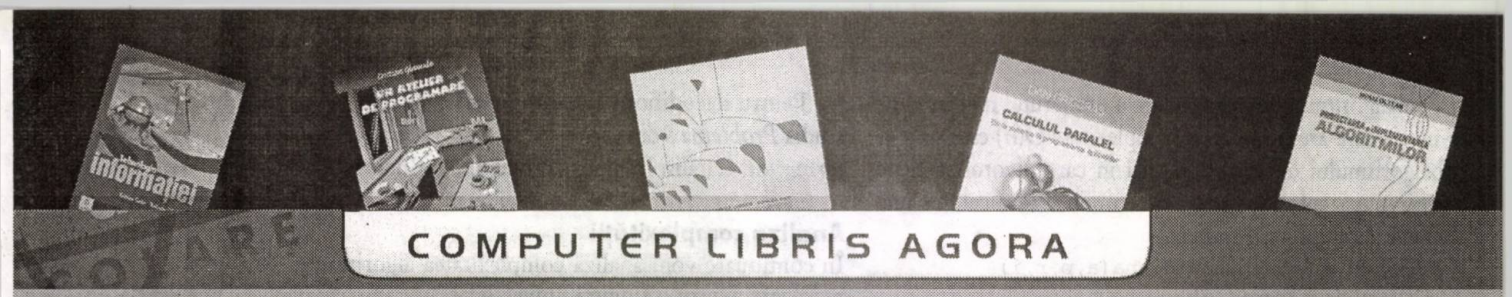
Operația de citire a datelor de intrare are ordinul de complexitate $O(n^2)$.

Operația de determinare a numărului de apariții a cheii s în matricea a are ordinul de complexitate $O(n^2 \cdot k)$, unde k reprezintă lungimea lui s.

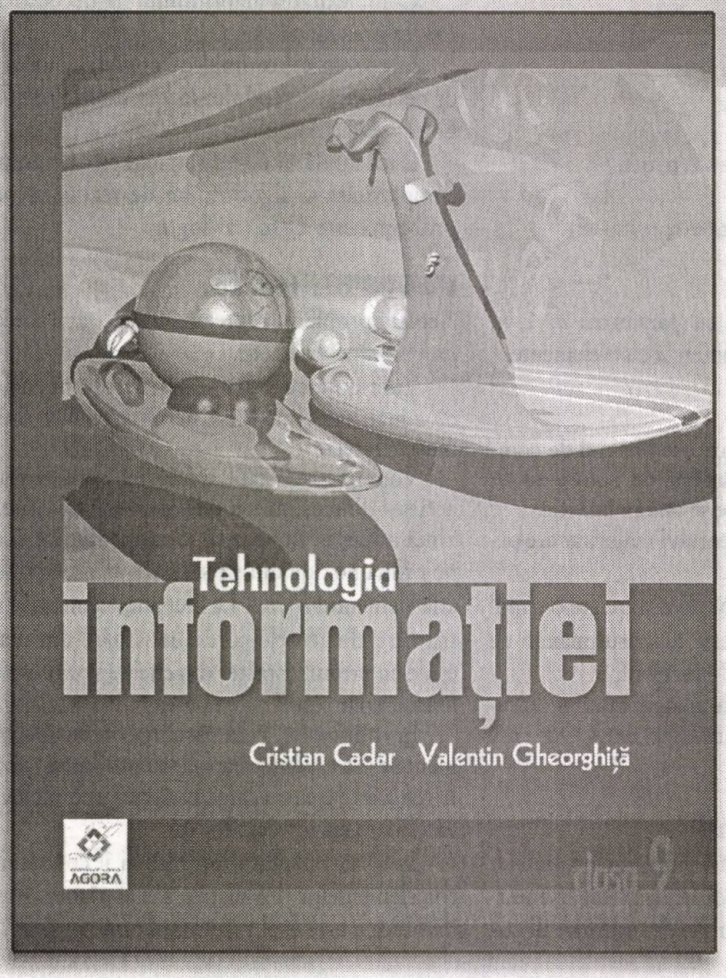
Operațiile de scriere a rezultatului are ordinul de complexitate $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(n^2) + O(n^2 \cdot k) + O(1) = O(n^2 \cdot k)$.

Silvestru-Cosmin Negruțeri este student în anul III la Universitatea Babeș-Bolyai din Cluj-Napoca și poate fi contactat prin e-mail la adresa **kosmin_2000@yahoo.com**.



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS
AGORA

16-19 octombrie 2003, București

ACM

În perioada 16-19 octombrie a avut loc la București ediția 2003/2004 a concursului internațional de programare ACM, la care au participat echipe reprezentând țări din sud-estul Europei. În paginile următoare vă prezentăm enunțurile problemelor propuse spre rezolvare la acest concurs.

Facultatea de Automatizări și Calculatoare din cadrul Universității "Politehnica" București a găzduit și ediția din acest a faie locale a concursului *ACM*, la care au participat echipe reprezentând țări din *sud-estul Europei*.

Pentru acest concurs s-au înscris 60 de echipe din *Bulgaria*, *Bosnia* și *Herțegovina*, *Grecia*, *Macedonia*, *Moldova*, *România*, *Ucraina* și *Iugoslavia*.

Dintre cele 60 de echipe au fost acceptate pentru concurs 53 de echipe de la 44 de universități din cele opt țări.

Echipele au avut la dispoziție cinci ore pentru a rezolva nouă probleme, folosind unul din limbajele de programare ***C/C++***, ***Pascal*** sau ***Java***. Gradul de dificultate al problemelor a fost relativ ridicat.

Calculatoarele puse la dispoziția participanților au avut procesoare *Pentium IV* cu frecvența de 2,6 *GHz*, 256 *MB* de memorie *RAM* și *harddisk-uri* cu o capacitate de 60 *GB*. Acestea au fost interconectate într-o rețea. Sistemele de operare puse la dispoziție au fost Windows *XP* și *Linux*.

Trei dintre echipele participante au reușit să rezolve 7 probleme, trei echipe au rezolvat 6 probleme și trei echipe au rezolvat 5 probleme. Majoritatea echipelor au rezolvat cel mult 3 probleme.

Primul loc a fost obținut de echipa a doua de la *Universitatea "St. Kliment Ohridski"* din *Sofia*, *Bulgaria*. Echipa l-a avut ca îndrumător pe *Krassimir Manev* și a fost formată din *Kristiyan Haralambiev*, *Miloslav Sredkov* și *Petko Minkov*. Aceștia au reușit să rezolve șapte probleme cu o penalizare de 827 de puncte.

Locul al doilea a fost obținut de echipa a treia de la aceeași universitate și care a avut același îndrumător. Echipa a fost formată din *Anev Ivan*, *Riskov Ivaylo* și *Tsvetan Bogdanov*. Aceștia au reușit să rezolve tot șapte probleme, dar cu o penalizare de 922 de puncte.

Locul al treilea a fost obținut de echipa "*UNIBUC-COLEGIU*" de la *Universitatea București*. Echipa l-a avut ca îndrumător pe *Cristian Grozea* și a fost formată din *Angel-Stelian Proorocu*, *Matei Gruber* și *Tiberiu Dănet*. Și ei au rezolvat șapte probleme, dar penalizarea a fost de 1085 de puncte.

Primele trei echipe, membrii juriului precum și cei care au avut grijă de buna desfășurare a concursului au primit premii de la *IBM*.

În continuare vă prezentăm enunțurile celor nouă probleme propuse spre rezolvare la acest concurs.

P070301: Codificare delta

Un algoritm simplu de criptare care reprezintă o îmbunătățire substanțială adusă metodei substituției este metoda *delta*. În continuare prezentăm modul de funcționare al acestuia.

Literelor alfabetului englezesc cuprinse între *A* și *Z* le sunt asociate numere de la 0 la 25. Valorile numerice nu sunt atașate literelor în ordinea în care apar în alfabet, dar fiecare literă trebuie să fie asociată cu un singur număr.

Metoda de codificare delta implicită asociază literelor numerele în aceeași ordine în care apar în alfabet (ordinea lexicografică), deci *A* este asociat cu 1, *B* cu 2 etc. Această asociere este utilizată atunci când nu există o alta mai bună.

Asocierea literelor cu numere cuprinse între 0 și 25 poartă numele de *cifru*.

Pentru a codifica un mesaj se folosesc *diferențele* dintre valorile succesive asociate literelor din alfabet.

Pentru codificarea unui mesaj cu *metoda delta* trebuie respectate următoarele reguli:

- toate simbolurile, în afară de litere, sunt transferate în mesajul criptat neschimbate;
- când o literă este precedată de un simbol care nu este literă sau nu este precedată de nici un simbol (cazul în care aceasta este prima literă din mesajul inițial) se consideră că este precedată de litera *A*;
- dacă o literă este majusculă, atunci va fi majusculă și în mesajul criptat, iar dacă nu este majusculă, nu va fi majusculă nici în mesajul criptat;
- când o literă este precedată de o altă literă, atunci valoarea diferenței este egală cu valoarea asociată primei litere care se scade din valoarea asociată celei de-a doua. Litera care are asociată valoarea diferenței modulo 26 o va înlocui pe cea de-a doua în mesajul criptat.

Sarcina voastră este de a cripta și decripta mesaje folosind această metodă.

Date de intrare

Fișierul de intrare **A.IN** conține pe fiecare linie câte o comandă urmată de parametrii corespunzători și fișierul se termină cu *EOF (End Of File - sfârșit de fișier)*.

O comandă este constituită din unul dintre cuvintele: ENCRYPT, DECRYPT sau CIPHER urmat de cel puțin un spațiu. Modul în care sunt scrise aceste comenzi nu este important (de exemplu, encrypt și enCrypt desemnează aceeași comandă).

Comanda ENCRYPT primește ca parametru (este urmată de) textul care trebuie criptat folosind cifrul curent.

Comanda DECRYPT primește ca parametru (este urmată de) textul care trebuie decriptat folosind cifrul curent.

Pentru comenzile ENCRYPT și DECRYPT, dacă nu a fost furnizat un cifru, se va folosi cifrul implicit.

Comanda CIPHER trebuie să fie urmată de cele 26 de litere ale alfabetului. Ordinea în care se află aceste litere determină valorile care le vor fi asociate astfel că prima literă din acest șir va fi asociată cu valoare 0, a doua cu valoarea 1 etc. Literele pot fi separate între ele prin 0 sau mai multe spații.

În cazul în care cifrul furnizat nu este valid (de exemplu, nu apar toate literele din alfabet sau o literă apare de două ori) se va folosi cifrul implicit.

Date de ieșire

Rezultatele vor fi furnizate la ieșirea standard. Pentru fiecare linie din fișierul de intrare se va furniza o linie la ieșire astfel:

- în cazul în care o linie nu începe cu una din cele trei comenzi posibile urmată de cel puțin un spațiu, în fișierul de ieșire, pe linia corespunzătoare se va afișa mesajul "Command not understood";
- pentru comanda ENCRYPT, la ieșire, pe linia corespunzătoare se va fi scrie mesajul "RESULT: " urmat de textul mesajului criptat corespunzător celui primit ca parametru;
- pentru comanda DECRYPT, la ieșire, pe linia corespunzătoare se va fi scrie mesajul "RESULT: " urmat de textul mesajului decriptat corespunzător celui primit ca parametru;
- pentru comanda CIPHER, la ieșire, pe linia corespunzătoare se va fi scrie mesajul "Good cipher. Using " urmat de cifrul scris cu majuscule, dacă cifrul este valid și "Bad cipher. Using default." în caz contrar.

Restricție

- nici o linie nu conține mai mult de 10.000 de caractere.

Exemplu

A.IN

```
ENCRYPT The quick brown fox leaped over the lazy dog.
CIPHER qwertyuiopasdfghjklzxcvbnm
ENCRYPT The quick brown fox leaped over the lazy dog.
DECRYPT Xsf juwgv fuyzb rcd oaomza nguw xsf olpd evu.
ENCRIPT This won't work!
CIPHER qAZwSXedCRfv tgbyhnujmikolp
ENCRYPT The quick brown fox leaped over the lazy dog.
DECRYPT Vsh puwgb gibeb rgc ocilda kbiw vsh owag ehh.
CIPHER qwertyuopasdfghjklzxcvbnm
ENCRYPT The quick brown fox leaped over the lazy dog.
DECRYPT Tox qeoui bqxir fjj ltwppz ohjn tox lpzz dls.
```

Ieșirea standard

```
RESULT: Tox qeoui bqxir fjj ltwppz ohjn tox lpzz dls.
Good cipher. Using QWERTYUIOPASDFGHJKLZXCVBNM.
RESULT: Xsf juwgv fuyzb rcd oaomza nguw xsf olpd evu.
RESULT: The quick brown fox leaped over the lazy dog.
Command not understood.
Good cipher. Using QAZWSXEDCRFVTGBYHNUJMIKOLP.
RESULT: Vsh puwgb gibeb rgc ocilda kbiw vsh owag ehh.
RESULT: The quick brown fox leaped over the lazy dog.
Bad cipher. Using default.
RESULT: Tox qeoui bqxir fjj ltwppz ohjn tox lpzz dls.
RESULT: The quick brown fox leaped over the lazy dog.
```

P070302: Parola ascunsă

Câteodată programatorii au metode din cele mai diverse pentru a-și ascunde parolele. De exemplu, să vedem cum *Billy "Hacker" Geits* își ascunde propriile parole.

Billy își alege un șir de caractere de lungime *L* format din litere mici ale alfabetului latin. Pentru acest șir de caractere *Billy* face toate cele *L* - 1 deplasări circulare la stânga cu o poziție și le pune unele sub altele. Dintre aceste *L* - 1 șiruri astfel obținute, înaintea cărora se trece șirul inițial, se alege cel care este primul în ordine lexicografică, parola constituind-o un prefix al acestuia.

De exemplu, pentru șirul alabala, șirurile care se obțin prin deplasări circulare, incluzându-l și pe acesta, sunt:

```
alabala
labalaa
abalaal
balaala
alaalab
laalaba
aalabal
```

Șirul care este primul în ordine lexicografică se află pe poziția a șasea (numerotarea șirurilor se face începând cu poziția 0), și în concluzie, prima literă a acestui șir se află pe poziția a șasea din șirul inițial.

Scrieți un program care pentru un șir *S* dat determină poziția celei "mai mici"(primei) deplasări în ordine lexicografică. Dacă cel mai mic șir de caractere apare de mai multe ori, să se cere cea mai mică poziție pe care acesta începe.

Date de intrare

Fișierul de intrare **B.IN** conține pe prima linie numărul *T* al testelor.

Pe fiecare dintre următoarele *T* linii se află o pereche formată dintr-un număr întreg *L* și un șir de caractere *S* se-

parate printr-un singur spațiu. L reprezintă lungimea șirului de caractere.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Rezultatul constă în T linii.

Pentru fiecare linie din fișierul de intrare se va scrie pe o linie un singur număr care reprezintă poziția determinată, conform enunțului problemei, pentru un șir S de lungime L.

Rezultatele pentru fiecare test se vor scrie în ordinea în care apar în fișierul de intrare.

Restricție

- $5 \le L \le 100.000$.

Exemplu

`B.IN`	*Ieșirea standard*
2	1
6 baabaa	6
7 alabala	

P070303: Petrecerea nebunatică

La "petrecerea nebunatică" participă n persoane. Acestea sunt așezate în jurul unei mese. După fiecare minut o pereche de vecini își schimbă locurile între ei.

Sarcina voastră este să determinați timpul minim, exprimat în minute, după care participanții la petrecere stau în ordine inversă.

Date de intrare

Fișierul de intrare `C.IN` conține pe prima linie numărul de teste. În continuare, pe o linie separată, pentru fiecare test, se află numărul n de persoane care participă la "petrecerea nebunatică".

Date de ieșire

Rezultatele vor fi furnizate la ieșirea standard. Pentru fiecare test din fișierul de intrare se va scrie pe o singură linie numărul minim de minute după care participanții sunt așezați în ordine inversă. Rezultatele se vor scrie în ordinea în care testele apar în fișierul de intrare.

Restricție

- $1 \le N \le 32767$.

Exemplu

`C.IN`	*Ieșirea standard*
3	2
4	4
5	6
6	

P070304: Supermagazin

Un supermagazin are de vânzare o mulțime `Prod` de produse. Pentru fiecare produs `x` ∈ `Prod` vândut până la scurgerea a d_x unități întregi de timp, care sunt măsurate începând cu momentul începerii vânzării de produse, se obține un profit p_x. Vânzarea unui produs durează o unitate de timp.

O planificare a vânzărilor este o submulțime ordonată `Sell` ⊆ `Prod`, astfel încât oricare produs `x` ∈ `Sell` se vinde înainte de expirarea timpului d_x sau chiar în momentul d_x.

Profitul `Profit(Sell)` al unei planificări `Sell` a vânzărilor este egal cu suma profiturilor care se obțin din vânzarea tuturor produselor din planificare. O planificare optimă este o planificare pentru care profitul este maxim.

Să considerăm produsele `Prod` = {`a`, `b`, `c`, `d`} pentru care: $(p_a, d_a) = (50, 2)$, $(p_b, d_b) = (10, 1)$, $(p_c, d_c) = (20, 2)$, respectiv $(p_d, d_d) = (30, 1)$. Planificările și profiturile corespunzătoare care se pot obține pentru aceste produse sunt:

planificare	*profit*	*planificare*	*profit*
{a}	50	{a, c}	70
{b}	10	{c, a}	70
{c}	20	{b, c}	30
{d}	30	**{d, a}**	**80**
{b, a}	60	{d, c}	50

De exemplu, planificarea `Sell` = {`d`, `a`} ne indică faptul că vânzarea produsului `d` începe la momentul 0 și se termină la momentul 1, în timp ce vânzarea produsului `a` începe la momentul 1 și se încheie la momentul 2. Fiecare dintre cele două produse sunt vândute până la expirarea termenului limită. Această planificare este optimă și profitul său este 80.

Sarcina voastră este să scrieți un program care pentru o mulțime de produse determină profitul care se poate obține pentru o planificare optimă.

Date de intrare

Prima linie a fișierului de intrare `D.IN` este format din mai multe teste.

Fiecare test este format dintr-un număr natural n, care reprezintă numărul de produse din mulțime, urmat de n perechi (p_i, d_i), $1 \le i \le n$, care reprezintă profitul obținut în urma vinderii produsului p_i până la termenul limită d_i.

Testele apar în fișierul de intrare unele după altele. Datele din acest fișier pot fi separate între ele prin oricâte spații (incluzând sfârșitul de linie).

Datele de intrare se termină cu *EOF* și sunt corecte.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard, astfel pentru fiecare test în parte se va scrie pe o linie separată profitul planificării optime. Rezultatele pentru teste se scriu în ordinea în care testele apar în fișierul de intrare.

Restricții

- $0 \le n \le 10.000$;
- $1 \le p_i \le 10.000$, $1 \le i \le n$;
- $1 \le d_i \le 10.000$, $1 \le i \le n$.

Exemplu

`D.IN`

```
4  50 2  10 1   20 2   30 1

7  20 1   2 1   10 3   100 2 8 2   5 20   50 10
```

Ieșirea standard

```
80
185
```

Explicație

Primul test este chiar exemplul prezentat în enunț, deci profitul planificării optime este 80.

Pentru cel de-al doilea test avem 7 produse. Planificarea optimă conține toate produsele din P, mai puțin cele cu indicii 2 și 5, deci profitul planificării optime pentru acest set de date este 185.

P070305: Stăpânul inelului

Frodo are de îndeplinit o misiune nobilă, dar dificilă. El trebuie să distrugă un inel magic și blestemat. În această aventură el trebuie să treacă printr-un loc periculos numit *Mordor* și să arunce inelul într-o prăpastie în flăcări.

Frodo a plecat de acasă de o bună bucată de timp și, momentan, merge pe un drum drept și lung, presărat cu tufișuri din loc în loc. Fiind foarte obosit, el consideră că ar fi timpul să se odihnească. Singurul loc sigur în care poate poposi este un tufiș a cărui poziție poate fi calculată folosind o formulă magică. Această formulă folosește valoarea lui P care reprezintă produsul distanțelor dintre perechile de tufișuri adiacente care se află de-a lungul drumului.

Frodo cunoaște numai distanțele dintre fiecare pereche de tufișuri și formula magică, dar, din păcate, nu cunoaște valoarea lui P. Puteți să-l ajutați să determine această valoare?

Date de intrare

Fișierul de intrare `E.IN` conține mai multe seturi de date.

Fiecare set de date este format dintr-un număr n, care reprezintă numărul de distanțe, urmat de n distanțe în ordine crescătoare.

Numerele din fișierul de intrare sunt separate prin oricâte spații (incluzând sfârșitul de linie).

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Pentru fiecare set de date trebuie scrisă pe o linie separată valoarea calculată pentru P, dacă aceasta se poate calcula sau textul "`No solution`" în caz contrar. Rezultatele pentru fiecare test se vor scrie în ordinea în care testele apar în fișierul de intrare.

Restricții și precizări

- numărul de tufișuri care se află de-a lungul drumului este cel puțin doi și cel mult 1.000;
- $1 \le P \le 1.000.000.000$.

Exemplu

`E.IN`

```
6
1 2  2   3  3 5
3
1 2 2
```

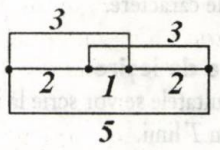


Figura 1

Ieșirea standard

```
4
No solution
```

Explicație

În figura 1 este ilustrată poziția tufișurilor pentru primul set de date.

Pentru cel de-al doilea set de date pozițiile tufișurilor care se află de-a lungul drumului nu pot fi deduse din valorile datelor de intrare, deci distanțele dintre punctele adiacente nu pot fi calculate.

P070306: Subsecvența comună

O subsecvență a unei secvențe date este chiar subsecvența dată cu mai puține elemente (posibil nici unul).

Fie o secvența $X = (x_1, x_2, ..., x_m)$. O secvență $Z = (z_1, z_2, ..., z_k)$ este subsecvență a lui X dacă există o secvență strict crescătoare $(i_1, i_2, ..., i_k)$ de indici din X, astfel încât pentru $1 \le j \le k$ să avem $x_{i_j} = z_j$. De exemplu, secvența $Z = (a, b, f, c)$ este o subsecvență a lui $X = (a, b, c, f, b, c)$ cu secvența de indici $(1, 2, 4, 6)$.

Dându-se două secvențe X și Y, determinați valoarea lungimii celei mai lungi subsecvențe comune lui X și Y.

Date de intrare

Fișierului de intrare `F.IN` conține mai multe seturi de date. Datele pentru fiecare set constau în două șiruri de caractere care reprezintă cele două secvențe X și Y. Secvențele din fișierul de intrare sunt separate prin oricâte spații.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Pentru fiecare set de date, pe o linie separată, se va scrie lungimea subsecvenței comune maxime. Rezultatele pentru fiecare set de date se vor scrie în ordinea în care apar în fișierul de intrare.

Exemplu

`F.IN`		*Ieșirea standard*
`abcfbc`	`abfcab`	4
`programming`	`contest`	2
`abcd`	`mnp`	0

P070307: Rețeaua electrică

O rețea electrică este formată din mai multe noduri (producători, distribuitori și consumatori) conectate între ele prin linii de transport.

Într-o rețea electrică există cel puțin un producător.

Un nod u poate fi aprovizionat cu o cantitate $s(u) \geq 0$ de curent electric, poate produce $0 \leq p(u) \leq p_{max}(u)$ curent electric, poate consuma $0 \leq c(u) \leq \min(s(u), c_{max}(u))$ curent și poate distribui $d(u) = s(u) + p(u) - c(u)$.

Se aplică următoarele restricții:

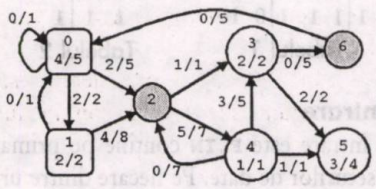
- $c(u) = 0$ pentru orice producător,
- $p(u) = 0$ pentru orice consumator și
- $c(u) = 0$ și $p(u) = 0$ pentru orice distribuitor.

Între două noduri u și v există cel mult o linie de transport. Aceasta poate transporta $0 \leq l(u, v) \leq l_{max}(u, v)$ curent electric de la nodul u la nodul v.

Fie $Con = \sum_u c(u)$ cantitatea de curent consumată în rețea. Problema cere să se determine cea mai mare valoare a lui *Con* care se poate atinge.

În figura următoare este prezentat un exemplu de rețea electrică.

u	tip	$s(u)$	$p(u)$	$c(u)$	$d(u)$
0	producător	0	4	0	4
1	producător	2	2	0	4
3	consumator	4	0	2	2
4	consumator	5	0	1	4
5	consumator	3	0	3	0
2	distribuitor	6	0	0	6
6	distribuitor	0	0	0	0



Eticheta x/y care apare în dreptul unui producător u indică faptul că $p(u) = x$ și $p_{max}(u) = y$. Eticheta x/y care apare în dreptul unui consumator u indică faptul că $c(u) = x$ și $c_{max}(u) = y$. Eticheta x/y care apare în dreptul unei linii de transport (u, v) indică faptul că $l(u, v) = x$ și $l_{max}(u, v) = y$. Puterea consumată de această rețea este $Con = 6$. Se observă că mai există și alte stări ale rețelei, dar valoarea *Con* nu poate depăși valoarea 6.

Date de intrare

Numele fișierului de intrare este **G.IN**. Acesta conține mai multe seturi de date. Fiecare set de date codifică o rețea electrică.

Un set de date începe cu patru numere naturale n, n_p, n_c, m care reprezintă numărul total de noduri, numărul de producători, numărul de consumatori, respectiv numărul de linii de transport. Cele patru numere sunt urmate de m triplete $(u,v)z$ care indică faptul că linia de transport care leagă nodurile u și v are capacitatea maximă $l_{max}(u, v)$ egală cu z. Nodurile sunt numerotate începând cu 0. Cele m triplete sunt urmate de n_p perechi $(u)z$ care reprezintă faptul că producătorul u poate produce $p_{max}(u) = z$ curent.

Setul de date se încheie cu n_c perechi $(u)z$ care reprezintă faptul că un consumator u poate consuma $c_{max}(u) = z$ curent electric.

Datele din fișierul de intrare sunt numere întregi și sunt separate prin oricâte spații în afară de tripletele și perechile, care nu conțin spații.

Datele de intrare sunt corecte și se termină cu *EOF*.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Pentru fiecare set de date, pe o linie separată, se va scrie valoarea maximă obținută pentru *Con*.

Restricții

- $0 \leq n \leq 100$;
- $0 \leq n_p \leq n$;
- $1 \leq m \leq n^2$;
- $0 \leq c_{max}(u), p_{max}(u) \leq 10.000$, $0 \leq u < n$;
- $0 \leq l_{max}(u, v) \leq 10.000$, $0 \leq u, v < n$.

Exemplu

H.IN

```
2 1 1 2 (0,1)20 (1,0)10 (0)15 (1)20
7 2 3 13 (0,0)1 (0,1)2 (0,2)5 (1,0)1 (1,2)8
   (2,3)1 (2,4)7 (3,5)2 (3,6)5 (4,2)7 (4,3)5
   (4,5)1 (6,0)5 (0)5 (1)2 (3)2 (4)1 (5)4
```

Ieșirea standard

```
15
6
```

Explicație

Primul set de date codifică o rețea electrică cu două noduri și două linii de transport. Producător este nodul 0 și $p_{max}(0) = 15$, iar consumator este nodul 1 și $c_{max}(1) = 20$. Pentru cele două linii avem $l_{max}(0, 1) = 20$ și $l_{max}(1, 0) = 10$. Valoarea maximă pentru *Con* este 15.

Cel de-al doilea set de date codifică rețeaua din exemplul din enunț, deci valoarea maximă pentru *Con* este 6.

P070308: Pompierii

Un matematician își păstra toate documentele într-un dulap situat în apropierea biroului său. Într-o zi a izbucnit un incendiu care i-a distrus o parte din documente. Din fericire, câteva dintre ecuațiile pe care le-a rezolvat de-a lungul carierei sale erau doar parțial deteriorate. Fiecare ecuație era formată dintr-o expresie în partea stângă și rezultatul în partea dreaptă. Expresiile parțial deteriorate constau din toate numerele și parantezele, dar din păcate, unele dintre semnele aritmetice nu se mai puteau distinge din cauza focului. O altă problemă era că rezultatele ecuațiilor au fost împrăștiate și matematicianul nu era sigur dacă un anumit răspuns corespundea rezultatului unei anumite ecuații.

Sarcina voastră este să-l ajutați pe matematician să afle dacă expresiile și rezultatele pe care a reușit să le salveze se potrivesc.

Pentru a-l putea ajuta, aveți la dispoziție o expresie care conține numere întregi cuprinse între 1 și 999, semnele matematice simple (+, -, ·, /), parantezele și semnul de întrebare care înlocuiește semnele pierdute.

Pentru fiecare expresie dată singura voastră sarcină este să spuneți dacă o expresie poate sau nu poate avea rezultatul cerut.

Pentru a vă ajuta, matematicianul a ales numai expresii care au următoarele restricții:

- expresiile nu conțin mai mult de 100 de simboluri;
- parantezele nu conțin decât cel mult o operație matematică și doi operanzi, dar oricare dintre cei doi operanzi sau amândoi pot avea forma unei expresii între paranteze;
- constantele prezente în expresii nu au semn (expresiile nu conțin numere negative);
- numărul maxim de semne de întrebare din expresii este mai mic sau egal cu 10.

Calcularea rezultatului ar trebui să se facă după următoarele reguli:

- operatorii · și / au prioritate mai mare decât + și -, iar parantezele pot schimba prioritatea;
- operatorii +, -, · și / sunt asociativi la stânga, ceea ce înseamnă că se grupează de la stânga la dreapta, astfel, dacă a, b și c sunt numere avem: $a \cdot b \cdot c = (a \cdot b) \cdot c$, $a / b / c = (a / b) / c$, $a / b \cdot c = (a / b) \cdot c$, $a + b + c = (a + b) + c$, $a - b + c = (a - b) + c$ etc.
- când se împart două numere întregi, trebuie să ignorați partea fracțională, de exemplu 2 / 5 = 0, 9 / 5 = 1, 100 / 6 = 16.

Date de intrare

Prima linie a fișierului de intrare **H.IN** conține un singur număr N care reprezintă numărul de ecuații prezente în fișier. Următoarele $2 \cdot N$ linii conțin ecuațiile.

O ecuație este definită pe două linii. Prima linie conține partea stângă a ecuației, iar cea de-a doua rezultatul (partea dreaptă a ecuației) dat sub forma unui număr întreg.

Fișierul de intrare nu conține spații, iar expresiile ecuațiilor nu au erori de sintaxă.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Pentru fiecare ecuație, pe o linie separată, se va scrie cuvântul "yes" dacă ecuația poate avea valoarea cerută și cuvântul "no" în caz contrar.

Exemplu

H.IN	*Ieșirea standard*
3	yes
1?((2*(3*4))+(5+6))	no
35	no
1?2*3+4-14	
0	
1?3*4/5*6+12	
11	

P070309: Polinoame binare

O funcție f definită pe mulțimea $\{0, 1\}^n$ de vectori binari și care ia valori din mulțimea $\{0, 1\}$ poartă numele de *funcție booleană de n variabile* și se notează cu $f(x_n, x_{n-1}, ..., x_1)$. Pentru criptografie sunt importante câteva proprietăți ale funcțiilor *booleene*.

Notăm cu $B(n, k)$ mulțimea vectorilor binari n-dimensionali care au k valori egale cu 1.

Pentru o funcție dată f se cere să se determine numărul de vectori de forma $(b_n, b_{n-1}, ..., b_1)$ din $B(n, k)$ pentru care $f(b_n, b_{n-1}, ..., b_1) = 1$.

O funcție *booleană* este unic determinată de valorile polinomului asociat modulo 2.

În aceste polinoame se folosesc doar operațiile de adunare și înmulțire modulo 2, după cum indică tabelul 1.

În polinomul asociat unei funcții orice produs de m variabile $x_{i1}, x_{i2}, ..., x_{im}$ poate să participe sau poate să nu participe. În general, forma acestor polinoame este $a_0 + a_1x_1 + a_2x_2 + a_3x_2x_1 + a_4x_3 + a_5x_3x_1 + ... + a_Nx_nx_{n-1}...x_1$, unde coeficienții a_j, $1 \le j \le N = 2^{n-1}$, pot fi 0 sau 1, iar dacă sunt 0, atunci produsele respective pot fi ignorate.

De exemplu, funcția *booleană* de disjuncție reprezentată în tabelul 2 este $0 + 1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_1 \cdot x_2 = x_1 + x_2 + x_1 \cdot x_2$.

+	0	1
0	0	1
1	1	1

*	0	1
0	0	0
1	0	1

Tabelul 1

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

Tabelul 2

Date de intrare

Fișierul de intrare este **F.IN** conține pe prima linie numărul T al seturilor de date. Pe fiecare dintre următoarele T linii se află câte un set de date.

Un set de date este format din două numere n și k separate printr-un singur spațiu. Aceste numere sunt urmate de un șir de 2^n cifre 0 și 1 care reprezintă coeficienții polinomului.

Date de ieșire

Rezultatele se vor scrie la ieșirea standard. Pentru fiecare set de date, pe o linie separată, se va scrie numărul de vectori determinați. Rezultatele pentru fiecare set se vor scrie în ordinea în care apar în fișierul de intrare.

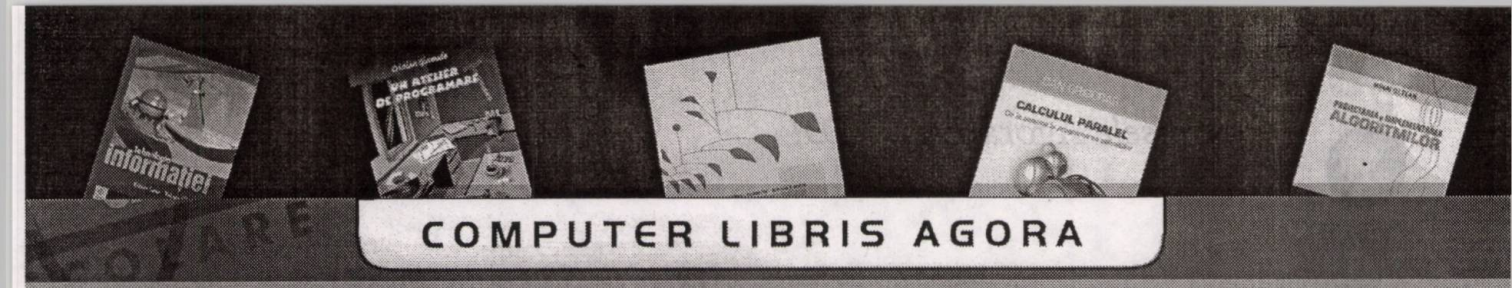
Exemplu

I.IN

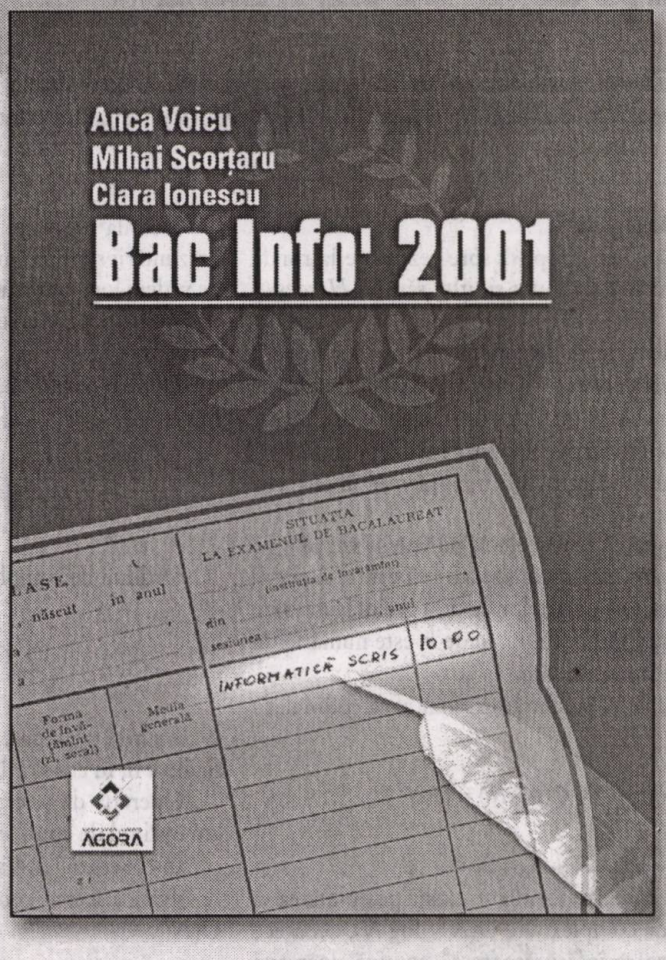
```
3
2 1 0111
4 2 1000000000000000
5 3 00000000000000000000000000000001
```

Ieșirea standard

```
2
6
0
```



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Etapa FINALĂ

În continuare vom prezenta soluțiile problemelor propuse spre rezolvare la etapa finală a ediției 2002/2003 a concursului de programare Bursele Agora.

P060314: Dragoni

Această problemă a fost propusă spre rezolvare la runda 17 a concursului având alt enunț și titlul său era *Heroes of Might & Magic*.

Această problemă se reduce la determinarea succesivă a unor înfășurători convexe. Vom determina o înfășurătoare pentru toate cele N puncte, apoi o alta pentru punctele rămase după eliminarea celor care se află pe prima înfășurătoare și vom continua până în momentul în care vor fi eliminate toate punctele.

Datorită faptului că toate cele N puncte se vor afla pe exact o înfășurătoare, este recomandabil să utilizăm *potrivirea Jarvis*. Determina punctelor de pe a i-a înfășurătoare ordinul se realizează în $O(N \cdot h_i)$, unde h_i este numărul de puncte de pe a i-a înfășurătoare.

Pentru toate cele K înfășurători vom avea ordinul de complexitate:

$$\sum_{i=1}^{K} O(N \cdot h_i) = O(N \cdot N) = O(N^2),$$

deoarece avem $h_1 + h_2 + ... + h_K = N$.

Dacă utilizăm *scanarea Graham*, există posibilitatea să obținem ordinul de complexitate $O(N^2 \cdot \log N)$, deoarece este posibil ca fiecare înfășurătoare să conțină doar trei puncte, deci putem avea până la $[N / 3] + 1$ înfășurători și pentru fiecare dintre acestea ordinul de complexitate este $O(N \cdot \log N)$.

Pe măsura determinării înfășurătorilor vom păstra pentru fiecare punct în parte numărul de ordine al înfășurătorii din care face parte începând cu 1. După determinarea tuturor înfășurătorilor aceste numere de ordine vor fi scrise în fișierul de ieșire.

Analiza complexității

Citirea datelor de intrare implică citirea celor N perechi de coordonate ale punctelor de observație, așadar ordinul de complexitate al operației de citire a datelor de intrare este $O(N)$.

Pentru cele N puncte vom determina o serie de înfășurători convexe, operație al cărei ordin de complexitate este $O(N^2)$, dacă se utilizează *potrivirea Jarvis*.

Metoda prin care a fost estimat acest ordin (în cadrul prezentării soluției) nu este corectă în întregime din punct de vedere logic. Practic, pentru a i-a înfășurătoare numărul de puncte pentru care aceasta este determinată nu este N, ci $N - h_1 - h_2 - ... - h_{i-1}$.

Obținem astfel, pentru a i-a înfășurătoare următoarea formulă pentru calculul ordinului de complexitate:

$$O\left(\left(N - \sum_{j=1}^{i-1} h_j\right) \cdot h_i\right).$$

Ordinul de complexitate al întregii operații ar deveni:

$$\sum_{i=1}^{K} O\left(\left(N - \sum_{j=1}^{i-1} h_j\right) \cdot h_i\right).$$

Cu toate acestea, dacă efectuăm calculele, rezultatul final va fi, în cazul cel mai defavorabil, tot $O(N^2)$.

Operația de scriere a rezultatelor în fișierul de ieșire are ordinul de complexitate $O(N)$.

În concluzie, ordinul de complexitate a algoritmului de rezolvare a acestei probleme este $O(N) + O(N^2) + O(N) = O(N^2)$.

P060315: Joc

Aceasta este o problemă *NP* iar 15 dintre testele cu care s-au verificat soluțiile au fost generate aleator.

Punctajele obținute de concurenți la această problemă nu au variat foarte mult, dar, după cum era de așteptat, au fost distincte. Problema a avut ca scop departajarea concurenților și a fost folosit un astfel de tip de problemă pentru a fi în concordanță cu rundele de tipul $4 \cdot k + 2$.

Această problemă poate fi rezolvată folosind metoda *backtracking*, dar nici pentru date de test foarte mici, o astfel de rezolvare nu s-ar fi încadrat în timp. O rezolvare *backtracking* pentru această problemă este foarte greu de implementat.

Problema a fost propusă spre rezolvare la un concurs de programare în *PHP*. Aceasta avea numele *Pegs*, enunțul original poate fi găsit pe *site*-ul **www.codewalkers.com** și diferă de cea propusă la etapa finală prin dimensiunea tablei

n care este tot timpul 7 și timpul de execuție este de 30 de secunde.

Problema se poate rezolva folosind un algoritm euristic care să încerce la fiecare pas stabilirea mutării optime.

Un astfel de algoritm are ordinul de complexitate $O(n)$, dă rezultate suficient de bune comparativ cu cele furnizate de concurenți, poate fi îmbunătățit folosind diverse strategii și pot fi căutate soluții până în momentul anterior expirării timpului de execuție.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(m)$, unde m este numărul de scobituri care conțin bile.

Datorită faptului că nu cunoaștem nici o rezolvare care să se încadreze în timpul de execuție, putem spune că ordinul de complexitate pentru abordarea *backtracking* este $6 \cdot O(k!)$, unde constanta 6 reprezintă numărul maxim de mutări care se pot executa pentru a ocupa o scobitură liberă, iar k reprezintă numărul mediu de mutări care se pot executa pentru o anumită configurație. Determinarea valorii k este destul de dificilă.

Ordinul de complexitate al operației de scriere a rezultatelor este $O(m)$.

În concluzie, ordinul de complexitate a algoritmului de rezolvare a acestei probleme este $O(m) + O(m!) + O(m) = O(m!)$.

P060316: Produse

Pentru a rezolva această problemă considerăm p_k o permutare a mulțimii $\{1, 2, ..., N\}$.

Produsul asociat acestei permutări, conform enunțului problemei, este egal cu $a_{1,pk1} \cdot a_{2,pk2} \cdot ... \cdot a_{N,pkN}$, unde p_{kl} reprezintă valoarea celui de-al l-lea element al permutării p_k. Acest produs este în continuare egal cu $1^2 \cdot p_{k1} \cdot 2^2 \cdot p_{k2} \cdot ... \cdot N^2 \cdot p_{kN}$, care este egal cu $N!^2 \cdot N! = N!^3$.

Datorită faptului că avem $N!$ permutări distincte, vom avea $N!$ produse distincte, deci produsul al cărui număr de cifre trebuie calculat este egal cu $N!^3 \cdot N! = N!^4$.

În acest moment am ajuns la problema determinării numărului de cifre al lui $N!^4$.

Pentru a determina numărul de cifre al lui $N!^4$ pornim de la observația că orice număr natural K poate fi scris sub forma $K = 10^{\lg K}$, de unde rezultă că numărul de cifre al lui K este egal cu $[\lg K] + 1$. Folosind această observație știm că numărul de cifre al lui $N!^4$ este egal cu $[\lg N!^4] + 1 = 4 \cdot [\lg N!] + 1$.

Conform formulei lui *Stirling* de determinare a valorii $\ln N!$ obținem următoarea aproximare:

$$\ln N! \approx \left(N + \frac{1}{2}\right) \cdot \ln N - N + \frac{1}{2} \cdot \ln(2 \cdot \pi \cdot N) + \frac{1}{12 \cdot N} - \frac{1}{360 \cdot N^3}$$

În final, rezultatul problemei este dat de expresia:

$$[4 \cdot \ln N!/\ln 10] + 1.$$

Problema determinării numărului de cifre al lui $n!$ a devenit deja o problemă clasică, ea apărând pentru prima dată la *Olimpiada Balcanică de Informatică* din 1997, unde a fost propusă de dl. prof. dr. *Adrian Atanasiu*. Această problemă a fost propusă spre rezolvare și la runda a 2-a a concursului de programare *Bursele Agora* ediția 1999/2000.

Analiza complexității

Ordinul de complexitate a operației de citire a datelor de intrare este $O(1)$.

Ordinul de complexitate a operației de determinare a numărului de cifre al lui $N!^4$ este $O(1)$.

Ordinul de complexitate a operației de scriere a rezultatelor este $O(1)$.

În concluzie, ordinul de complexitate a algoritmului de rezolvare pentru această problemă este $O(1) + O(1) + O(1) = O(1)$.

P060317: Mijloc

Pentru a rezolva această problemă vom împărți, cel puțin teoretic, șirul inițial care are $2 \cdot N$ elemente în două șiruri a și b care au N elemente și sunt sortate crescător.

Datorită faptului că șirul inițial are lungimea $2 \cdot N$ se observă că există două elemente care, dacă sortăm șirul, se află la mijlocului lui.

Pentru a determina poziția pe care se află unul dintre elementele mediane ale șirului inițial (elementul pentru care există, în șirul inițial, N elemente mai mari sau egale cu el și N elemente mai mici sau egale cu el) vom folosi algoritmul descris în continuare.

La fiecare pas considerăm elementele i și j care se află la mijlocul celor două șiruri a și b. Dacă $i > j$ atunci vom elimina din șirul a toate elementele care se află înaintea elementului i și din șirul b toate elementele care preced elementul j, altfel vom elimina din șirul a toate elementele care preced elementul i și din șirul b toate elementele care se află înaintea elementului j.

Acest algoritm se încheie în momentul în care șirurile a și b au un singur element. Unul dintre aceste două elemente este soluția problemei. Cele două elemente care rămân nu reprezintă cele două mediane ale șirului inițial decât dacă nu se află în aceeași jumătate a acestuia.

Algoritmul prezentat efectuează $\log N$ apeluri ale funcției `IsGreater` furnizată de bibliotecă și cele două apeluri permise ale procedurii `Result`.

Analiza complexității

Pentru această problemă datele sunt citite de către biblioteca folosită.

Ordinul de complexitate a operației de determinare a unui element median este $O(\log N)$.

Ordinul de complexitate a operației de furnizare a rezultatului este $O(1)$.

În concluzie, ordinul de complexitate a algoritmului de rezolvare pentru această problemă este $O(1) + O(\log N) = O(\log N)$.

ALGORITMI - CALCULABILITATE

Horia Georgescu

În cadrul acestui articol vom aborda problema existenței algoritmilor de rezolvare pentru diferite probleme și vom insista asupra calculabilității funcțiilor.

În cei 12 ani de existență, în *Gazeta de Informatică* (apoi *GInfo*) au apărut nenumărate articole privind algoritmii, sub toate aspectele lor. Dar toate aceste articole au avut un numitor comun și anume au presupus existența algoritmilor pentru problemele studiate.

Există algoritmi pentru orice problemă?

Această problemă a stat în atenția matematicienilor chiar înainte de apariția calculatoarelor și face obiectul *teoriei calculabilității*. La bază stau cercetările din al patrulea deceniu al secolului trecut ale unor personalități precum *Church*, *Gödel*, *Kleene*, *Post* și *Turing*. Ei au propus diferite definiții ale noțiunii de algoritm, bazate fie pe limbaje de programare, fie pe mașini matematice, definiții care s-au dovedit a fi echivalente.

Articolul de față își propune să prezinte cititorilor elementele de bază ale teoriei calculabilității, cu scopul de a dovedi că există probleme pentru care nu pot fi elaborați algoritmi. Prezentarea nu este originală, ci urmează întocmai o parte din minunata carte *Computability, Complexity and Languages, Academic Press* 1983, a lui ***Martin Davis*** și ***Elaine Weyuker***.

Ne exprimăm speranța că cititorii ***GInfo*** vor fi interesați de acest subiect, care din păcate lipsește chiar din programa multor facultăți cu profilul informatică.

Limbajul de programare S

O primă definiție a noțiunii de algoritm se bazează pe limbajul *S*. Într-o primă etapă se va face prezentarea acestui limbaj foarte simplu, urmând ca apoi să căpătăm încredere în posibilitățile sale.

Un program *P* scris în limbajul *S* este o secvență de instrucțiuni (un număr finit de instrucțiuni scrise într-o anumită ordine), program în care intervin:

- numere naturale;
- variabile de intrare, notate de obicei prin x_1, x_2, ...;
- variabila de ieșire y;
- variabile intermediare (de lucru), notate de obicei prin z_1, z_2, ...

cu mențiunea că valorile luate de variabile pot fi orice numere naturale.

Deoarece lucrăm numai cu numere naturale, vom spune, uneori, pur și simplu număr în loc de număr natural.

Înainte de a prezenta instrucțiunile limbajului *S*, precizăm că ele pot fi precedate, opțional, de etichete. Acestea sunt reprezentate prin litere mari (eventual indexate cu numere naturale), cuprinse între paranteze drepte și plasate la începutul instrucțiunii. Se admite ca mai multe instrucțiuni să aibă aceeași etichetă; rațiunea acestui fapt va fi prezentată în alt articol.

Instrucțiunile admise de limbajul *S* și semnificația lor sunt următoarele:

- $v \leftarrow v + 1$ — valoarea curentă a variabilei v crește cu o unitate;
- $v \leftarrow v - 1$ — valoarea curentă a variabilei v scade cu o unitate (dacă era strict pozitivă), respectiv rămâne 0 (dacă era 0);
- **if** $v \neq 0$ **goto** L — dacă valoarea curentă a variabilei v este nenulă, atunci se face transfer necondiționat la *prima* instrucțiune din program etichetată cu L; dacă nici o in-

strucțiune nu este etichetată cu L, atunci programul se termină; dacă valoarea curentă a lui v este 0, atunci se trece la instrucțiunea următoare;
- $v \leftarrow v$ — este instrucțiunea cu efect nul, a cărei utilitate va apărea ulterior.

Exemplul 1

Programul:

```
[A] x ← x - 1
    y ← y + 1
    if x ≠ 0 goto A
```

se termină pentru orice valoare inițială α a lui x. La ieșire vom avea:

$$y = \begin{cases} 1, \alpha = 0 \\ \alpha, \alpha > 0 \end{cases}$$

Exemplul 2

Programul:

```
[A] x ← x + 1
    if x ≠ 0 goto A
```

nu se termină niciodată, indiferent de valoarea inițială a lui x.

Observație

Faptul că programele în limbajul S produc o singură valoare de ieșire nu este o restricție. Într-adevăr, dacă dorim să obținem mai multe valori de ieșire, vom scrie câte un program pentru fiecare și vom executa succesiv aceste programe. Menționăm că, în acest studiu, suntem interesați *numai de existența algoritmilor, nu și de eficiența lor.*

Fie P un program scris în limbajul S. Fie V mulțimea tuturor variabilelor (de intrare, de lucru și de ieșire) care apar în P. Vom da în continuare o definiție riguroasă a *semanticii* programului P.

Definim ***starea*** programului la un moment oarecare a executării sale ca fiind o funcție s: $V \to N$, unde N este mulțimea numerelor naturale. Deci, o stare este dată de valorile curente ale variabilelor din program.

O *configurație* a programului P de lungime n (având n instrucțiuni) este o pereche (i, s) cu $i \in \{1, 2, ..., n, n+1\}$ și s stare.

O *configurație inițială* are forma $(1, s_0)$, unde:
- $s_0(u) = r_i$, dacă $u = x_i$ și r_i este valoarea inițială a lui x_i și
- $s_0(u) = 0$, dacă u nu este o variabilă de intrare.

O *configurație terminală* are forma $(n + 1, s)$. Vom asimila transferul la o etichetă inexistentă cu transferul la instrucțiunea cu numărul $n + 1$.

Fie (i, s) o configurație neterminală. Succesoarea ei (j, t) este definită astfel:

1) Dacă instrucțiunea i este $v \leftarrow v + 1$, atunci $j = i + 1$, iar:
   - $t(u) = u$, dacă $u \neq v$ și
   - $t(u) = u + 1$, dacă $u = v$.
2) Dacă instrucțiunea i este $v \leftarrow v - 1$, atunci $j = i + 1$, iar:
   - $t(u) = s(v) - 1$, dacă $u = v$ și $s(v) > 0$ și
   - $t(u) = s(v)$, în caz contrar.
3) Dacă instrucțiunea i este $v \leftarrow v$, atunci $j = i + 1$, și $t = s$.
4) Dacă instrucțiunea i este `if` $v \neq 0$ `goto` L, atunci $t = s$, iar:
   - $j = i + 1$, $s(v) = 0$;
   - $j = n + 1$, $s(v) > 0$ și nu există instrucțiuni etichetate cu L, și
   - $j = k$, dacă $s(v) > 0$ și instrucțiunea k este prima instrucțiune cu eticheta L.

Se observă că succesoarea oricărei configurații neterminale este unic determinată și nu depinde de eventuala etichetare a sa.

Un *calcul* al programului P este o *secvență de configurații consecutive* $c_0, c_1, ..., c_k$ cu c_k configurație finală (terminală).

În continuare vom lucra cu funcții parțiale (parțial definite). Mai precis, pentru f: $N^m \to N$ vom nota prin *Dom f* elementele din N^m pentru care funcția f este definită.

Fie P un program în care variabilele de intrare sunt $x_1, x_2, ..., x_m$. Atunci *funcția (parțială) calculată de programul* P este $\psi_P^{(m)}: N^m \to N$ definită astfel:

1) $\psi_P^{(m)}(x_1, x_2, ..., x_m) = y'$ dacă există un calcul $c_0, c_1, ..., c_k$ al lui P cu $c_0 = (1, s_0)$ și $s_k(y) = y'$;
2) $\psi_P^{(m)}(x_1, x_2, ..., x_m) = \uparrow$ (nedefinit) dacă există un șir infinit de configurații $c_0, c_1, ...$ al lui P cu $c_0 = (1, s_0)$, deci dacă programul P nu se termină.

Observație

Fie P un program în care apar variabilele de intrare $x_1, x_2, ..., x_m$. Pentru orice număr natural n, îi putem asocia o funcție $\psi_P^{(n)}$ astfel:
- dacă $n \leq m$, atunci $\psi_P^{(n)}(x_1, x_2, ..., x_n) = \psi_P^{(m)}(x_1, x_2, ..., x_m)$, adică $x_{n+1}, ..., x_m$ sunt ignorate;
- dacă $n > m$, atunci $\psi_P^{(n)}(x_1, ..., x_n) = \psi_P^{(m)}(x_1, ..., x_m, 0, ..., 0)$.

Fie f: $N^m \to N$. Funcția f se numește ***parțial calculabilă*** dacă există un program P în limbajul S cu $\psi_P^{(m)} = f$. Dacă *Dom* $f = N^m$, atunci f se numește ***calculabilă.***

Exemplul 1 arată că funcția f: $N \to N$ dată de:

$$f(x) = \begin{cases} 1, x = 0 \\ x, x > 0 \end{cases}, \text{ este } \textit{calculabilă.}$$

Exemplul 2 arată că funcția unară cu domeniul vid este *parțial calculabilă.*

Macroinstrucțiuni

Această secțiune urmărește să căpătăm încredere în limbajul S. Vom arăta că principalele prelucrări uzuale asupra numerelor naturale pot fi realizate prin programe scrise în acest limbaj.

Macroinstrucțiunile sunt abrevieri pentru secvențe de instrucțiuni în limbajul S. Menirea lor este de a scrie programe cât mai inteligibile. Prezența unei macroinstrucțiuni

într-un program trebuie interpretată ca prezența în acel punct al programului a unei *dezvoltări* ale sale.

Chiar dacă în același program apare în diferite locuri o aceeași macroinstrucțiune, aceasta nu înseamnă că va fi inserată aceeași dezvoltare a sa. Mai mult, chiar impunem ca aparițiile unei aceleiași macroinstrucțiuni pe poziții diferite din program să presupună dezvoltări diferite ale macroinstrucțiunii (variabile de lucru și etichete distincte). De asemenea trebuie respectată regula ca valoarea variabilelor care apar în macroinstrucțiune, cu excepția celor care apar în membrul stâng al unei atribuiri, să nu fie modificate ca efect al executării macroinstrucțiunii.

Macroinstrucțiunea **goto** L are următoarea dezvoltare posibilă:

```
[A] z ← z + 1
    if z ≠ 0 goto A
```

Macroinstrucțiunea $v \leftarrow 0$ are dezvoltarea:

```
[A] v ← v - 1
    if v ≠ 0 goto A
```

unde eticheta A nu mai apare nicăieri în programul care conține această macroinstrucțiune.

3) Macroinstrucțiunea $v \leftarrow k$ are dezvoltarea:

```
v ← 0
v ← v + 1
...
v ← v + 1
```

unde instrucțiunea $v \leftarrow v + 1$ apare de exact k ori.

Exemplul 3

Funcția $f: N \to N$ dată de $f(x) = x$ este *calculabilă*.

Pentru demonstrație, vom scrie următorul program P cu $\psi_P^{(1)} = f$, program care va asigura păstrarea valorii inițiale a lui x:

```
[A] if x ≠ 0 goto B
    goto C
[B] x ← x - 1
    y ← y + 1
    z ← z + 1
    goto A
[C] if z ≠ 0 goto D
    goto E
[D] z ← z - 1
    x ← x + 1
    goto C
```

unde primele 6 instrucțiuni copiază valoarea lui x în variabilele y și z, iar următoarele recopiază valoarea lui z în x.

Putem acum introduce macroinstrucțiunea $v \leftarrow u$ cu dezvoltarea:

```
    v ← 0
[A] if u ≠ 0 goto B
    goto C
[B] u ← u - 1
    v ← v + 1
    z ← z + 1
    goto A
[C] if z ≠ 0 goto D
    goto E
[D] z ← z - 1
    u ← u + 1
    goto C
[E] v ← v
```

care "transcrie" funcția din exemplul 3.

Se impun următoarele remarci:

- nu am inițializat pe z cu 0, deoarece este clar că după orice executare a macroinstrucțiunii, valoarea sa va fi 0;
- apare necesitatea instrucțiunii $v \leftarrow v$;
- ca de obicei, variabila z nu mai apare nicăieri în program (este o variabilă de lucru "nouă"), iar etichetele A, B, C, D, E nu mai apar nicăieri în program (sunt etichete "noi"). Având convingerea că cititorul s-a obișnuit deja cu aceste reguli, nu le vom mai aminti în continuare, presupunându-le subînțelese.

Exemplul 4

Funcția $f: N \times N \to N$ definită prin $f(x_1, x_2) = x_1 + x_2$ este *calculabilă*.

Un program care calculează această funcție este următorul:

```
    y ← x1
    z ← x2
[B] if z ≠ 0 goto A
    goto E
[A] z ← z - 1
    y ← y + 1
    goto B
```

iar macroinstrucțiunea $z \leftarrow z_1 + z_2$ are următoarea dezvoltare posibilă:

```
    u ← z1
    v ← z2
[B] if v ≠ 0 goto A
    goto E
[A] v ← v - 1
    u ← u + 1
    goto B
[E] z ← u
```

Exemplul 5

Funcția $f: N \times N \to N$ definită prin $f(x_1, x_2) = x_1 \cdot x_2$ este *calculabilă*.

Un program care calculează această funcție este următorul:

```
    z ← x2
[B] if z ≠ 0 goto A
    goto E
[A] z ← z - 1
```

```
    y ← y + x1
    goto B
```

iar macroinstrucțiunea $z \leftarrow z_1 \cdot z_2$ are următoarea dezvoltare posibilă:

```
    u ← 0
    v ← z2
[B] if v ≠ 0 goto A
    goto E
[A] v ← v - 1
    u ← u + z1
    goto B
[E] z ← u
```

Exemplul 6

Fie $f: N \times N \to N$ funcția parțială definită prin:

$$f(x_1, x_2) = \begin{cases} x_1 - x_2, x_1 \geq x_2 \\ \uparrow, x_1 < x_2 \end{cases}.$$

Un program care calculează această funcție este următorul:

```
    y ← x1
    z ← x2
[C] if z ≠ 0 goto A
    goto E
[A] if y = 0 goto B
    goto A    {y = 0 și z = 0}
[B] y ← y - 1
    z ← z - 1
    goto C
```

Macroinstrucțiunea asociată este $z \leftarrow z_1 - z_2$ și are următoarea dezvoltare posibilă:

```
    u1 ← z1
    u2 ← z2
[C] if u2 ≠ 0 goto A
    goto E
[A] if u1 = 0 goto B
    goto A    {y = 0 și z = 0}
[B] u1 ← u1 - 1
    u2 ← u2 - 1
    goto C
[E] z ← u1
```

Exemplul 7

Fie $f: N^m \to N$ o funcție (parțial) calculabilă, calculată de un program P. Vom introduce macroinstrucțiunea:

$$z \leftarrow f(z_1, \ldots, z_m).$$

Pentru a obține o dezvoltare a acestei macroinstrucțiuni, să presupunem că în programul P variabilele de intrare sunt $x_1, \ldots, x_m$, variabilele de lucru sunt $z_1, \ldots, z_k$, iar etichetele sunt $A_1, \ldots, A_p$, unde (cu eventuala excepție a lui A_p), orice etichetă care apare într-o instrucțiune **if** este eticheta a cel puțin uneia dintre instrucțiunile din P. Atunci o dezvoltare a macroinstrucțiunii $z \leftarrow f(z_1, \ldots, z_m)$ este următoarea:

```
    y' ← 0
    x1' ← x1
    ...
    xm' ← xm
    z1' ← 0
    ...
    zk' ← 0
    P'
    I
```

unde:

- $y', x'_1, \ldots, x'_m, z'_1, \ldots, z'_k$ sunt variabile de lucru noi;
- $A'_1, \ldots, A'_p$ sunt etichete noi;
- P' se obține din P înlocuind pe $y, x_1, \ldots, x_m, z_1, \ldots, z_k, A_1, \ldots, A_p$ cu $y', x'_1, \ldots, x'_m, z'_1, \ldots, z'_k, A'_1, \ldots, A'_p$.
- I este una dintre instrucțiunile:
  - `z ← y'`, dacă eticheta A_p este definită;
  - `[A'p] z ← y'`, dacă eticheta A_p nu este definită.

Se observă imediat că dacă pentru anumite valori inițiale $r_1, \ldots, r_m$ ale lui $x_1, \ldots, x_m$ programul P nu se termină, atunci nici executarea macroinstrucțiunii $z \leftarrow f(z_1, \ldots, z_m)$ nu se termină pentru $z_1, \ldots, z_m$ având înainte de executarea macroinstrucțiunii valorile $r_1, \ldots, r_m$.

Dacă o macroinstrucțiune dintr-un program nu se termină, atunci nici programul care o folosește nu se termină.

Numim predicat o funcție $f: N^m \to \{0, 1\}$, parțial sau total definită, unde 0 corespunde falsului, iar 1 corespunde adevărului.

Exemplul 8

Fie $P: N^m \to \{0, 1\}$ un predicat calculabil. Atunci putem introduce macroinstrucțiunea:

```
if P(z1, ..., zm) goto L
```

a cărei dezvoltare este:

```
z ← P(z1, ..., zm)
if z = 0 goto L
```

Un caz particular îl constituie predicatul:

$$P(x) = \begin{cases} 1, x = 0 \\ 0, x \neq 0 \end{cases},$$

care este calculabil conform programului:

```
if x ≠ 0 goto E
y ← y + 1
```

ceea ce permite introducerea următoarei macroinstrucțiuni:

```
if v = 0 goto L.
```

Va urma...

În următoarele articole care constituie continuarea celui prezentat vom interacționa cu câteva probleme nedecidabile și vom vedea cum problema de față poate fi tratată folosind alte modele de abordare.

Dl. prof. dr. Horia Georgescu este cadru didactic la Universitatea București, fondator și director științific al GInfo și poate fi contactat prin e-mail la adresa g_horia@hotmail.com.

Metode de REDUCERE a COMPLEXITĂȚII

Mihai Stroe

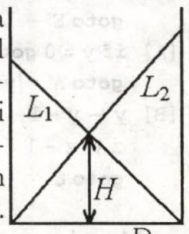
În cadrul acestui articol vom prezenta câteva tehnici cu ajutorul cărora se poate face trecerea, pentru anumite tipuri de probleme, de la complexitatea liniară a unor pași din rezolvare la complexitatea logaritmică, folosind principiul căutării binare.

În ultimii ani au început să apară multe probleme care se pot rezolva folosind algoritmi al căror ordin de complexitate este $O(N \cdot \log N)$ sau $O(N^2 \cdot \log N)$. Câteva rezolvări ale unor probleme clasice de acest gen (de exemplu, problema determinării subșirului crescător de lungime maximă) sunt cunoscute. Alte rezolvări se bazează pe structuri de date speciale (*heap*-uri, arbori *AVL*, arbori de intervale etc.)

O categorie de probleme este considerată mai aparte deoarece, la un moment dat, în rezolvarea lor se introduce căutarea binară. Vom examina câteva astfel de probleme. Cititorul este invitat să încerce să rezolve problemele înainte de a citi sugestiile autorului.

Enunțurile problemelor pe care le vom examina sunt următoarele:

- Se consideră un șir de N numere naturale și un număr natural S. Se cere să se aleagă trei numere din acest șir, astfel încât suma lor să fie egală cu S.
- Se consideră un labirint sub forma unei matrice pătratice de dimensiune N ale cărei elemente pot lua valorile 0 sau 1, un punct de plecare și un punct de sosire. Dacă un element din matrice are valoarea 1, înseamnă că pe poziția corespunzătoare din labirint este zid, sau spațiu liber în caz contrar. Se cere să se afișeze dimensiunea celui mai mare pătrat care poate fi deplasat de la punctul de plecare la punctul de sosire. Un pătrat de latura K va ocupa în fiecare moment K^2 poziții din matrice. Pătratul se poate deplasa pe orizontală și verticală și nu poate trece prin ziduri.
- Se consideră un graf neorientat. Fiecare muchie are asociate o lungime și un grad de periculozitate. Se cere să se afișeze un drum de la nodul 1 la nodul N, astfel încât gradul de periculozitate al celei mai periculoase muchii să fie cât mai mic posibil; dacă există mai multe drumuri, se va afișa drumul care are cea mai mică lungime.
- Se consideră un arbore cu N noduri ($N \leq 100.000$). În cadrul său se pot alege K baze, situate în K noduri. Pentru fiecare nod se calculează distanța până la cea mai apropiată bază. Determinați o modalitate de alegere a bazelor, astfel încât cea mai mare dintre distanțele nodurilor față de cea mai apropiata bază să fie cât mai mică posibil.
- Doi pereți verticali se află la o distanță D unul de celălalt. O scândură de lungime L_1 pleacă de la baza primului perete și se sprijină pe al doilea. O scândură de lungime L_2 pleacă de la baza celui de-al doilea perete și se sprijină pe primul. Cele două scânduri se ating la înălțimea H, într-un punct situat undeva între cei doi pereți. Dându-se L_1, L_2 și H, să se calculeze D; este permisă o eroare de 10^{-5}.



Prima problemă poate fi rezolvată prin metoda "*forței brute*" folosind trei cicluri repetitive `for`. Din nefericire, valoarea pentru N s-ar putea să fie prea mare pentru ca această rezolvare să se încadreze în timp.

Soluția propusă care folosește căutarea binară este următoarea: se sortează șirul, se folosesc două cicluri repetitive `for` pentru a alege două numere (fie acestea N_1 și N_2), apoi se caută binar valoarea $S - N_1 - N_2$. Această rezolvare are ordinul de complexitate $O(N^2 \cdot 2 \log N)$ și mai poate fi optimizată (de exemplu, dacă valoarea N_1 depășește valoarea S se iese din primul ciclu etc.). Varianta optimizată va avea șanse mari să treacă toate testele.

Invităm cititorul să găsească o rezolvare în $O(N^2)$, fără a folosi căutarea binară.

Cea de-a doua problemă este mai dificilă.

Dacă pătratul ar avea latura 1, folosirea algoritmului lui *Lee* ar rezolva problema. Acest algoritm va fi folosit oricum, dar dificultatea constă, la fiecare pas, în evitarea atingerii unui zid. Dacă se deplasează un pătrat de latură K, nu se dorește examinarea a K poziții vecine pentru, să zicem, o deplasare la dreapta. Aceasta se poate rezolva

folosind sume parțiale pe linii și pe coloane; o altă modalitate constă în precalcularea, pentru fiecare poziție, a laturii maxime a unui pătrat care poate fi deplasat din poziția respectivă pe fiecare din cele patru direcții.

Până în acest punct, algoritmul constă în încercarea deplasării unui pătrat de latură 1, 2, 3 etc., până când deplasarea nu mai reușește; se alege cea mai mare dimensiune pentru care deplasarea a reușit. O îmbunătățire ar putea consta în folosirea căutării binare pentru lungimea laturii. Astfel, inițial se va încerca deplasarea unui pătrat de dimensiune $N/2$; dacă aceasta a reușit se va încerca deplasarea unui pătrat de dimensiune $3 \cdot N/4$, altfel se va testa un pătrat de dimensiune $N/4$ etc. Se va executa algoritmul lui *Lee* de $\log N$ ori, ceea ce va conduce la un ordin de complexitate $O(N^2 \cdot \log N)$.

În cazul celei de-a treia probleme, căutarea binară se va face în funcție de valorile gradelor de periculozitate; pentru fiecare alegere a unei astfel de valori se va executa algoritmul lui *Dijkstra* pentru a determina un drum de lungime minimă, folosind numai muchii cu gradul de periculozitate mai mic sau egal cu valoarea aleasă. Ordinul de complexitate al acestei soluții este tot $O(N^2 \cdot \log N)$.

Cea de-a patra problemă cere alegerea a K noduri ale arborelui. Dacă am face abstracție de limita pentru numărul de noduri, problema, astfel pusă, ar părea, la prima vedere, *NP* completă.

Există o variantă abordabilă a acestei probleme, în care se cunoaște distanța cerută D (cea mai mare distanță față de cea mai apropiată bază) și se cere determinarea numărului minim de baze. Putem rezolva această nouă problemă pornind de la nodurile cu un singur vecin, mergând spre "interiorul" arborelui și amplasând baze în noduri cât mai depărtate de nodurile din exterior, dar totuși suficient de aproape de ele, astfel încât distanța respectivă să nu depășească D. După construirea arborelui într-o formă ușor de exploatat, acest algoritm ar avea ordinul de complexitate $O(N)$. Observăm că, pentru o distanță D mică, numărul de baze ales va fi mare și reciproc. Distanța minimă cerută este limitată de faptul că trebuie să ne încadrăm în limita de K baze.

Soluția constă în a alege, pe rând, valori pentru D folosind căutarea binară. Pentru fiecare dintre aceste valori se va rezolva varianta abordabilă a problemei în $O(N)$. Dacă se obțin mai mult de K baze, D trebuie mărit; în caz contrar, D va fi micșorat.

Complexitatea acestei rezolvări este $O(N \cdot \log N)$.

Rezolvările pentru problemele a doua, a treia și a patra conduc la următoarea regulă generală: *dacă trebuie obținută o valoare maximă astfel încât să se respecte anumite condiții, și respectarea condițiilor este evidentă pentru valori mai mici, atunci folosind căutarea binară și testarea condiției vom obține rezultatul în* $\log N$ *pași.* Regula poate fi adaptată și pentru problemele de minim.

În problemele a doua și a treia valoarea căutată este latura pătratului, respectiv gradul maxim de periculozitate pentru o muchie. Testarea condițiilor amintite mai sus implică rezolvarea unor subprobleme, folosind algoritmi precum *Lee* și *Dijkstra*; aceste subprobleme sunt clasice și, în concluzie, evidente.

Cea de-a patra problemă pare puțin mai complicată, deoarece trecerea la varianta abordabilă a problemei nu este evidentă; folosind principiile din acest articol, astfel de treceri pot fi detectate destul de ușor.

Cea de-a cincea problemă ar putea admite o rezolvare geometrică. Aceasta nu este însă evidentă (cel puțin autorul nu a reușit să o găsească în timp util). Pe de altă parte, se observă că înălțimea la care se ating scândurile crește dacă distanța căutată scade.

Putem reformula cerința problemei astfel: "*să se găsească cea mai mare valoare D astfel încât înălțimea la care se ating scândurile să nu fie mai mică decât H*". Evident, la limită se obține exact aceeași cerință. Avantajul este că, pentru această abordare, putem folosi căutarea binară în funcție de valoarea D. Cunoscând D, L_1 și L_2, înălțimea la care se ating scândurile se poate determina destul de ușor, folosind cunoștințe elementare de geometrie plană. Se verifică dacă aceasta este mai mare decât H; dacă da, D va fi crescut, în caz contrar va fi micșorat.

Pare puțin ciudat să rezolvăm o problemă de geometrie folosind căutarea binară a rezultatului, dar practica demonstrează că este foarte eficient.

Abordarea este foarte bună în timpul concursurilor, unde timpul este o resursă critică; implementarea este simplă (fapt foarte important în concurs; *încercați să nu greșiți tocmai căutarea binară!*), timpul de execuție este mic și se economisește timp de gândire.

O problemă care se poate rezolva folosind același principiu este problema ***Proc*** propusă spre rezolvare la *ONI* 2003 la clasele XI-XII.

Observație

O parte dintre problemele din acest articol admit algoritmi de rezolvare mai eficienți. Am prezentat rezolvarea lor prin metoda din acest articol pentru a evidenția valoarea acesteia; multe probleme aparent dificile devin abordabile, obținându-se algoritmi optimi sau aproape optimi, care, de obicei, se încadrează în timpul de execuție acordat.

Sugestie

Cititorul este invitat să continue exercițiul, căutând soluții mai eficiente, fără a folosi căutarea binară. Reamintim că, pentru probleme mai dificile, căutarea soluției optime ar putea necesita un timp de gândire important, greu de găsit în timpul concursului.

Mihai Stroe este masterand la Universitatea Politehnica din București și poate fi contactat prin e-mail la adresa **mihai_stroe@yahoo.com**.

Algoritmul lui KIRKPATRICK

Mircea Digulescu, Andrei Matei

Problema localizării punctelor este una dintre problemele pentru care, deși sunt foarte ușor de rezolvat de către ochiul uman, mult timp nu s-a găsit un algortim optim. În acest articol vom prezenta un algoritm care folosește un spațiu de memorie proporțional cu O(n) și răspunde la interogări în timp de O(n log n).

Enunțul problemei

Enunțul problemei localizării punctelor (în plan) este următorul:

"*Se consideră o împărțire (divizare) a planului în celule poligonale. Dându-se un punct din plan, se cere să se precizeze în care dintre celule se află acesta.*"

Este ciudat că, deși pare o problemă simplă pe care ochiul uman o poate rezolva în mod natural, a durat mult să se ajungă la un algoritm de rezolvare care ocupă un spațiu de memorie $O(n)$ și este capabil să răspundă la o interogare într-un timp $O(\log n)$.

Primul care a dezvoltat un astfel de algoritm a fost *David G. Kirkpatrick*.

Prezentarea algoritmului

Pentru a se putea răspunde eficient la interogări este necesar ca celulele poligonale să fie preprocesate.

Acest algoritm pleacă de la supoziția că divizarea planului este o triangulare. În plus, trebuie ca și exteriorul divizării să fie triunghi. Pentru cazul general, vom începe prin a triangula fiecare celulă a divizării și a marca fiecare triunghi rezultat cu eticheta celulei din care face parte. Pentru a adăuga "exteriorul", vom mai adăuga trei puncte distincte cu coordonate "infinite" (se poate demonstra foarte ușor că orice punct din plan se află în interiorul triunghiului format de cele trei puncte) și vom uni aceste puncte (a, b, c în figura 1) cu puncte de pe înfășurătoarea convexă a diviziunii.

Deși s-ar putea crede că triangulând adăugăm multe muchii noi, se poate demonstra ușor că, în cel mai rău caz, numărul de muchii crește cu un factor constant.

Se observă că, dacă găsim triunghiul care conține punctul, atunci cunoaștem și celula din divizarea inițială.

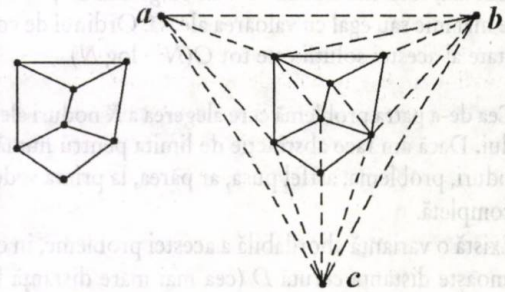


Figura 1: Triangularea unei diviziuni

Fie T_0 triangularea inițială. Un algoritm bazat pe *metoda lui Kirkpatrick* va produce o secvență de triangulări, T_0, T_1, T_2, ..., T_k (cu $k = O(\log n)$) astfel încât T_k constă într-un singur triunghi (cel format de cele trei puncte cu coordonate infinite din T_0) și fiecare triunghi din T_{i+1} se suprapune peste un număr constant de triunghiuri din T_i.

Vom vedea cum folosim o astfel de structură la interogări. În continuare construim această structură.

Presupunem că avem triangularea T_i și vrem să construim T_{i+1}. Pentru a fi siguri că acest proces se oprește după $O(\log n)$ pași ne vom asigura că numărul de triunghiuri din T_{i+1} este mai mic decât numărul de triunghiuri din T_i cu un factor constant. Fiecărei triangulări îi vom atașa un graf, astfel încât fiecare punct este un nod și există muchii între două noduri, dacă există segment între punctele corespunzătoare din triangulare. Din acest punct al rezolvării, prin T_i vom înțelege, în funcție de context, graful asociat triangulării T_i sau chiar triangularea.

La fiecare pas i vom alege o submulțime de noduri din T_i pe care le vom elimina împreună cu muchiile adiacente lor. După ce le eliminăm vom retriangula graful pentru a obține T_{i+1}. Nu se vor elimina niciodată punctele suplimentare cu coordonatele infinite.

Apare întrebarea "*cum alegem nodurile care se elimină din T_i astfel încât fiecare triunghi din T_{i+1} să se suprapună peste un număr constant de triunghiuri din T_i*".

Pentru a alege nodurile care vor fi eliminate vom ține cont de:

- ***grad maxim*** - fiecare nod pe care îl eliminăm să aibă gradul maxim egal cu o constantă d (să fie incident la maxim d muchii) și când eliminăm un astfel de nod, *gaura* rezultată va fi triangulată în T_{i+1} în maxim d - 2 triunghiuri; după retriangulare, fiecare triunghi nou se va suprapune peste maxim d triunghiuri din precedenta triangulare.
- ***set independent*** - între oricare două noduri care sunt eliminate să nu existe muchie (nodurile eliminate formează un set independent în T_i); această restricție ne va folosi la retriangulare; când eliminăm în acest fel m noduri diferite obținem m găuri disjuncte pe care le putem triangula independent; mai mult, deoarece fiecare gaură conține un număr constant de noduri, acestea se vor triangula în $O(1)$.

O problemă importantă este dacă vom reuși sau nu întotdeauna să găsim un set independent, suficient de mare cu grad limitat. Vrem ca mărimea acestui set să fie cel puțin o fracțiune constantă din numărul curent de noduri. S-a descoperit că există întotdeauna un astfel de set. Important este să alegem o valoare pentru d suficient de mare (dacă d este prea mic, nu vom găsi suficiente noduri). Vom demonstra că $d = 8$ este suficient de mare.

Lemă: *Dându-se un graf planar (cum este cazul triangulărilor noastre) cu n noduri, există un set independent de noduri cu grad maxim 8, cu cel puțin n/18 noduri. Acest set poate fi determinat într-un timp O(n).*

Vom arăta cum construim acest set și pe urmă vom demonstra mărimea sa:

marcăm toate nodurile care au gradul ≥ 9
cât timp există un nod nemarcat:
  alegem un astfel de nod v
  adăugăm v la set
  îl marcăm pe v și pe toți vecinii lui

Se observă că acest algoritm se termină în $O(n)$.

Demonstrația faptului că setul rezultat va avea cel puțin $n/18$ noduri este următoarea: o formulă fundamentală a lui *Euler* ne spune că într-un graf planar cu n noduri complet triangulat numărul e de muchii satisface relația $e = 3 \cdot n - 6$. Dacă adunăm gradele nodurilor, fiecare muchie este numărată de două ori.

Cum $\sum_v grad(v) = 2e = 6n - 12 < 6n$, o aproximare inferioară a mediei gradelor este 6. Vom demonstra în continuare că există cel puțin $n/2$ noduri cu gradul 8 sau mai mic. Să presupunem contrariul, și anume că există mai mult de $n/2$ noduri de grad 9 sau mai mare. Cum celelalte noduri trebuie să aibă gradul mai mare sau egal cu 3 (cu excepția celor trei noduri cu coordonatele infinite, adăugate artificial), rezultă că suma gradelor în graf este $9 \cdot n/2 + 3 \cdot n/2 + 6$, ceea ce contrazice ecuația de mai sus.

Când algoritmul de mai sus începe, cel puțin $n/2$ noduri sunt inițial nemarcate. De fiecare dată când selectăm un astfel de nod, din cauză că gradul său este 8 sau mai mic, marcăm cel mult 9 noduri noi (pe el însuși și cel mult 8 vecini). Deci acest pas poate fi repetat de cel puțin $(n/2)/9 = n/18$ ori, sau până când rămânem fără noduri nemarcate.

În acest moment demonstrația este completă.

Acum știm că putem construi un set convenabil cu cel puțin $n/18$ noduri. Numărul 18 pare destul de mare, dar în practică va fi mai mic. Totuși această constantă este un motiv pentru care algoritmul lui *Kirkpatrick* nu este folosit foarte des în practică.

Structura lui Kirkpatrick

Acum să vedem efectiv cum este construită structura de localizare (structura lui *Kirkpatrick*) . Începem cu T_0 și la fiecare pas selectăm un subset independent de noduri cu grad minim 8 pe care le eliminăm. Retriangulăm găurile obținute, cu observația că fiecare triunghi nou se suprapune peste maxim 8 triunghiuri din triangularea precedentă. Cu fiecare triangulare, numărul de noduri scade cu un factor de maxim 17/18. Pentru a ajunge la o triangulare alcătuită dintr-un singur triunghi, cel de la infinit, avem $(18 / 17) \cdot k = n$ sau $k = \log_{18/17} n \approx 12 \cdot \lg n$.

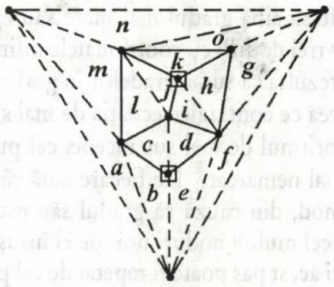
Structura de date se bazează pe aceste triangulări. Rădăcina structurii corespunde singurului triunghi din T_k. Nodurile de la următorul nivel sunt triunghiurile din T_{k-1}, urmate de cele din T_{k-2} și așa mai departe până la frunze, care sunt triunghiurile din T_0, triangularea inițială. Fiecare nod de pe nivelul i reține referințe la triunghiurile peste care se suprapune din nivelul $i + 1$ (sunt cel mult 8).

Pentru a găsi triunghiul în care se află un punct începem de la rădăcină. Dacă punctul nu se află în triunghiul asociat rădăcinii, am terminat; punctul nu se află în interiorul nici unui triunghi din triangularea inițială. Altfel, testăm dacă punctul e inclus în fiecare din cele cel mult 8 triunghiuri de pe nivelul următor care sunt incluse în triunghiul curent. Și așa mai departe până ajungem la frunze.

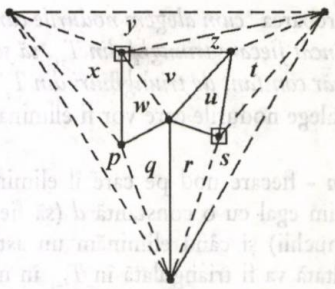
Construirea și analiza

Structura are $\log n$ niveluri și ne ia un timp constant să ne mutăm de la un nivel la următorul (maxim 8 teste de incluziune în triunghi), deci timpul unei interogări este $O(\log n)$. Dimensiunea structurii este egală cu suma mărimilor triangulărilor.

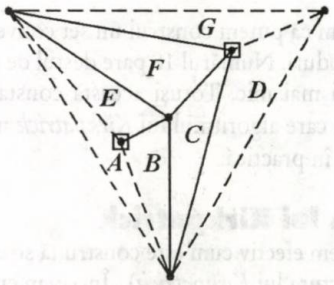
Cum numărul de triunghiuri al unei triangulări depinde liniar de numărul de puncte, rezultă că mărimea structurii este proporțională cu $n \cdot (1 + 17/18 + (17/18)^2 + (17/18)^3 + \ldots) \le 18 \cdot n$. Deci $O(n)$ înmulțită cu o constantă destul de mare.



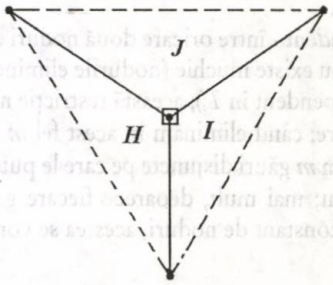
a) Triangularea inițială T_0



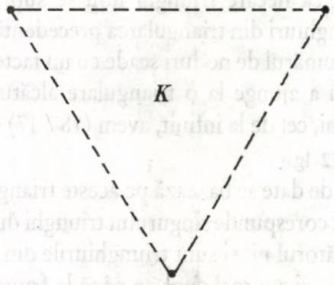
b) Triangularea T_1



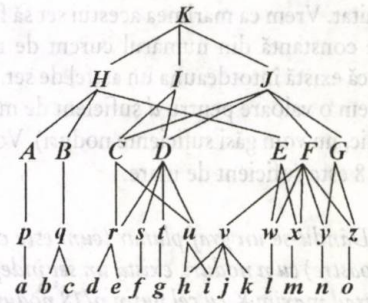
c) Triangularea T_2



d) Triangularea T_3



e) Triangularea T_5



f) Structura folosită pentru interogări

Figura 2: Construirea structurii folosită pentru interogări. Punctele încadrate sunt eliminate la pasul curent.

În figura 2 este prezentat modul în care se realizează construirea structurii care va fi folosită pentru interogări pentru exemplul din figura 1, prin eliminări de puncte și triangulări repetate.

Se poate observa că cea de-a cincea triangulare, și ultima de altfel, este echivalentă cu triunghiul format de punctele suplimentare pe care le-am adăugat.

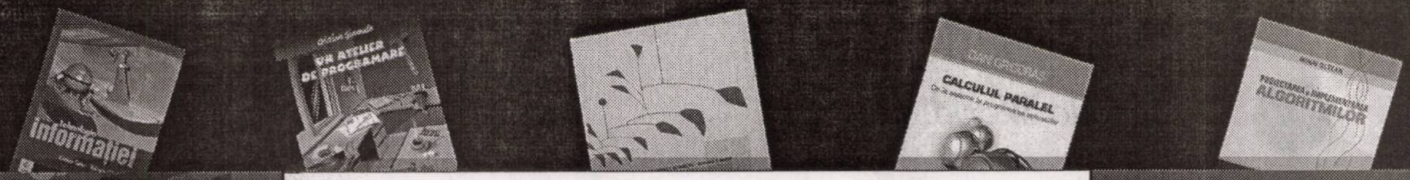
Dacă dorim să vedem în care celulă se află un anumit punct, atunci vom determina mai întâi triunghiul din T_1 din care face parte și apoi, în funcție de acest triunghi, vom determina celula (poligonul) din care acesta face parte.

În figura 2 f) se află structura pe care am construit-o și care va fi folosită la interogări pentru exemplul din figura 1. Fiecare nod din această structură reprezentând un triunghi, a fost conectat de nodurile, care reprezintă alte triunghiuri, peste care acesta se suprapune.

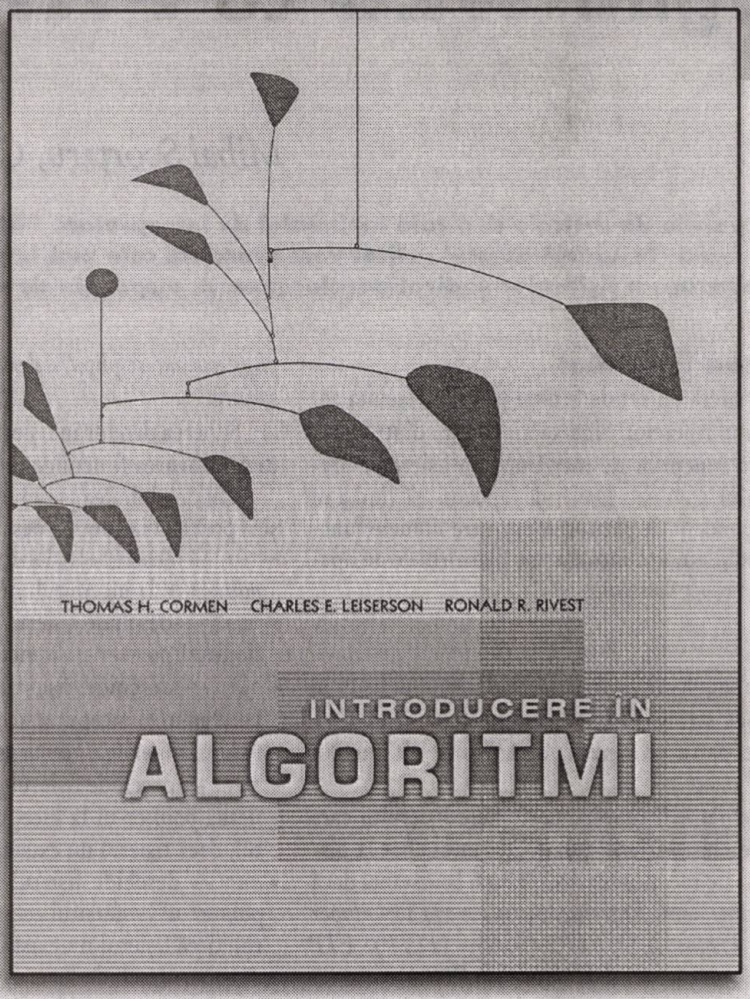
Bibliografie

1. David G. Kirkpatrick, *Optimal Search in Planar Subdivisions*, SIAM Journal on Computing, Vol. 12
2. Daniel Dominic Sleator, Robert Endre Tarjan, *A Data Structure for Dynamic Trees*, Symposium on Theory of Computation (1981), pp. 114-122
3. Franco P. Preparata, *A New Approach to Planar Point Location*, SIAM Journal on Computing, Vol. 10 (1981), Articolul 3 , pp. 473 - 482
4. Herbert Edelsbrunner, Lionidas J. Guibas, Jorge Stolfi, *Optimal point location in a monotone subdivision*, SIAM Journal on Computing, Vol. 15 (1986), Articolul 2, pp. 317 - 340
5. Michael Ian Shamos, *Geometric complexity*, In Conference Record of Seventh Annual ACM Symposium on Theory of Computation, pp. 224 - 233
6. Richard J. Lipton, Robert Endre Tarjan, *Application of a Planar Separator Theorem*, FOCS 1977, pp. 162-170

Mircea Digulescu și Andrei Matei sunt elevi în clasa a XII-a la Colegiul Național de Informatică Tudor Vianu din București și pot fi contactați prin e-mail la **mircea85@yahoo.com**, *respectiv* **andreimatei@home.ro**.



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

Reluăm seria de articole dedicate limbajului de interpretare PHP ajuns acum la versiunea 5. În cadrul acestui articol vom prezenta cele mai importante funcții de prelucrare a fișierelor și directoarelor puse la dispoziție de acesta.

Manipularea fișierelor

Limbajul *PHP* oferă un set de funcții pentru a facilita accesul la conținutul fișierelor. Sintaxa majorității dintre acestea este similară funcțiilor de manipulare a fișierelor prezente în limbajul *C/C++*. Datorită faptului că limbajul *PHP* este independent de platformă, printre numele funcțiilor se regăsesc și nume cunoscute pe platformele de operare **nix* și *Microsoft Windows*.

Funcția fopen

Această funcție se folosește pentru deschiderea unui fișier în vederea citirii sau scrierii datelor din/în acesta.

Funcția are patru parametri, dintre care ultimii doi sunt opționali.

Primul parametru îl reprezintă numele fișierului care urmează a fi deschis. Fișierul poate fi și unul aflat la distanță. Un fișier este recunoscut de *PHP* a fi la distanță dacă este accesibil printr-unul dintre protocoalele *HTTP* (*Hyper Text Transfer Protocol*), *HTTPS* (*Secure HTTP*), *FTP* (*File Transfer Protocol*) sau *FTPS* (*Secure FTP*).

Fișierele standard de intrare și ieșire pot fi accesate folosind următorii identificatori:

- *php://stdin* — fișierul standard de intrare pentru procesul curent creat de *PHP*;
- *php://stdout* — fișierul standard de ieșire pentru procesul curent creat de *PHP*;
- *php://stderr* — fișierul standard de erori pentru procesul curent creat de *PHP*;
- *php://input* — cu ajutorul acestui identificator se poate accesa conținutul datelor transmise paginii prin intermediul metodei *POST*;
- *php://output* — cu ajutorul acestui identificator se pot scrie date direct în pagina web.

Fișierele *php://input* și *php://stdin* pot fi deschise numai în scopul citirii, în timp ce fișierele *php://output*, *php://stdout* și *php://stderr* pot fi deschise numai pentru scriere.

Scrierea datelor în fișierul *php://output* are același efect ca și utilizarea funcțiilor `print` și `echo`.

Cel de-al doilea parametru este de tipul șir de caractere și reprezintă modul de deschidere a fișierului care poate fi:

- `"r"` — deschide fișierul pentru citire și poziționează *pointer*-ul la începutul acestuia;
- `"r+"` — deschide fișierul pentru citire și scriere și poziționează *pointer*-ul la începutul acestuia;
- `"w"` — deschide fișierul pentru scriere, poziționează *pointer*-ul la începutul acestuia și șterge conținutul; dacă fișierul nu există, acesta este creat.
- `"w+"` — deschide fișierul pentru citire și scriere, poziționează *pointer*-ul la începutul acestuia și șterge conținutul; dacă fișierul nu există, acesta este creat.
- `"a"` — deschide fișierul pentru scriere și poziționează *pointer*-ul la sfârșitul acestuia, însă fără a șterge conținutul; dacă fișierul nu există, acesta este creat.
- `"a+"` — deschide fișierul pentru citire și scriere și poziționează *pointer*-ul la sfârșitul acestuia, însă fără a șterge conținutul; dacă fișierul nu există, acesta este creat.

La sfâșitul șirului care reprezintă modul de deschidere a fișierului mai poate apărea și caracterul `"b"`. Acesta este util atunci când se dorește deschiderea fișierului în mod binar și nu în mod text. Pe sistemele de operare **nix* nu are nici un efect, în schimb pentru portabilitate este indicată folosirea lui.

Cel de-al treilea parametru este de tip întreg și poate avea valorile `0` sau `1` și indică funcției faptul că se poate folosi (valoarea 1) sau nu (valoarea 0) lista de directoare, cu care interpretorul a fost configurat, pentru a găsi fișierul care se dorește a fi deschis, în cazul în care numele acestuia nu include și o cale absolută.

Cel de-al patrulea parametru este de tipul `resource` și este folosit în scopul optimizării unor operații.

Funcția `fopen` returnează un număr întreg care reprezintă identificatorul de acces la fișierul care tocmai a fost deschis. Acest identificator va fi transmis tuturor funcțiilor pe care le veți utiliza pentru a accesa date din fișier.

În cazul în care deschiderea fișierului nu poate avea loc din diferite motive, atunci funția returnează valoarea logică **FALSE**.

De exemplu, pentru a deschide fișierul cu numele `"test.txt"` pentru a citi date, vom folosi următoarea linie de cod sursă: `$f = fopen("test.txt","r");`. Dacă dorim să scriem date în fișier, atunci îl putem deschide folosind următorul cod: `$f = fopen("test.txt","w");`.

Funcția tmpfile

Această funcție realizează crearea și deschiderea unui fișier temporar pentru scriere și citire. Funcția `tmpfile` nu are parametri și returnează un număr întreg care reprezintă identificatorul de acces al fișierului temporar.

Fișierul temporar va fi șters în mod automat în momentul în care este închis sau când *script*-ul își încheie execuția.

Apelul acestei funcții este identic cu cel al funcției **fopen** cu parametrii: un nume de fișier temporar (de obicei generat aleator) și `"w+b"` ca mod de deschidere a fișierului.

Funcția fclose

Această funcție realizează închiderea unui fișier care a fost deschis cu ajutorul funcțiilor `fopen` sau `tmpfile`.

Funcția `fclose` are un singur parametru de tip întreg, care reprezintă identificatorul de acces la fișierul care se dorește a fi închis.

Funcția returnează valoarea logică **TRUE** dacă închiderea fișierului a reușit și valoarea logică **FALSE** în caz contrar.

Un exemplu de apel al acestei funcții este `fclose($fd)`, unde `fd` reprezintă identificatorul de acces la fișierul care se dorește a fi închis.

Funcția feof

Această funcție verifică dacă s-a ajuns la sfârșitul fișierului sau nu.

Funcția `feof` are un singur parametru de tip întreg, care reprezintă identificatorul de acces la fișierul care se dorește a fi închis.

Funcția returnează valoarea logică **TRUE** dacă am ajuns la sfârșitul fișierului și valoarea logică **FALSE** în caz contrar.

Funcțiile fscanf

Această funcție citește date dintr-un fișier text și are doi parametri obligatorii.

Primul parametru este de tip întreg și reprezintă identificatorul de acces la fișierul din care se face citirea, iar cel de-al doilea parametru este de tipul șir de caractere și indică funcției formatul intrării. Formatul acestuia din urmă este similar formatului folosit pentru funcția `fscanf` prezentă în limbajul *C/C++*.

Ceilalți parametri reprezintă referințele la variabilele care vor conține rezultatul citirii.

Funcția returnează un vector cu valorile citite, deci prezența variabilelor nu mai este obligatorie ca în cazul limbajului *C/C++*.

De exemplu, dacă din fișierul `"c:\test.txt"` dorim să citim un număr și un șir de caractere separate printr-un spațiu, atunci vom folosi următorul cod:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fscanf
  </TITLE>
 </HEAD>
 <BODY>
    <?PHP
      $f = fopen("c:\\test.txt","r");
      $rez = fscanf($f,"%d %s\n");
      list($nr,$sir) = $rez;
      fclose($f);
   echo "Am citit numarul: ".$nr."<BR>";
   echo "si sirul: ".$sir;
  ?>
  </BODY>
</HTML>
```

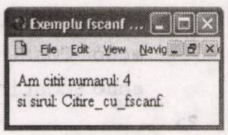


Figura 1: Funcția fscanf

În figura 1 poate fi observat efectul execuției acestui cod.

Funcția fread

Această funcție se folosește pentru citirea datelor din fișiere binare. Pentru compatibilitate cu toate sistemele de operare se recomandă ca la deschiderea fișierului, pentru citire, să se adauge caracterul `"b"` la parametrul care reprezintă modul de deschidere a fișierului.

Funcția `fread` are doi parametri obligatorii. Primul parametru este de tip întreg și reprezintă identificatorul de acces al fișierului din care se dorește să se citească date, iar al doilea parametru, tot de tip întreg, reprezintă numărul de octeți care se doresc a fi citiți.

Rezultatul returnat de această funcție este de tip șir de caractere care este format din octeții care s-au citit din fișierul de intrare.

În cazul în care în fișierul din care se citesc date nu mai este disponibil numărul de octeți specificat în cadrul celui de-al doilea parametru, atunci funcția returnează toți octeții disponibili de la poziția curentă din fișier și până la sfârșitul acestuia.

În continuare vă prezentăm un exemplu de folosire a acestei funcții, iar în figura 2 poate fi observat efectul execuției codului.

Codul este compatibil cu toate sistemele de operare pe care limbajul *PHP* poate fi instalat.

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fread
    </TITLE>
 </HEAD>
 <BODY>
   <?PHP
      $fis = "c:\\test.txt";
      $fd = fopen($fisr,"rb");
      $continut = fread($fd,filesize($fis));
      fclose($fd);
      echo "Continutul fisierului este: ";
      echo $continut;
  ?>
  </BODY>
</HTML>
```

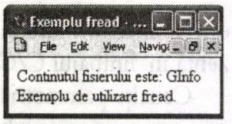


Figura 2: Funcția fread

Funcția fgetc

Această funcție citește următorul caracter disponibil din fișierul de intrare.

Funcția are un singur parametru de tip întreg care reprezintă identificatorul de acces la fișierul din care se dorește să se facă citirea.

Funcția `fgetc` returnează un șir de caractere, în cazul în care nu s-a ajuns la sfârșitul fișierului, format dintr-un singur caracter, și anume, cel citit din fișier.

În cazul în care ne aflăm la sfârșitul fișierului, funcția `fgetc` returnează valoarea logică **FALSE**.

În continuare vă prezentăm un exemplu de utilizare a acestei funcții și în figura 3 poate fi observat efectul execuției acestui cod:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fgetc
    </TITLE>
 </HEAD>
 <BODY>
   <?PHP
      $fisier = "c:\\test.txt";
      $fd = fopen($fisier,"r");
      $car = fgetc($fd);
      fclose($fd);
      echo "Primul caracter din fisier este: ";
      echo $car;
  ?>
  </BODY>
</HTML>
```

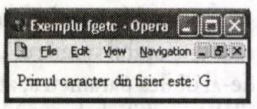


Figura 3: Funcția fgetc

Funcția fgets

Această funcție se folosește pentru a citi o linie dintr-un fișier de tip text. Sintaxa acestei funcții este similară cu cea a funcției `fread`, cu mențiunea că cel de-al doilea parametru este opțional.

Funcția `fgets` returnează un șir de caractere care conține datele citite. Citirea datelor din fișier se încheie în momentul în care s-a citit un număr de caractere egal cu valoarea celui de-al doilea parametru (în cazul în care acesta este prezent), în momentul întâlnirii unui sfârșit de linie sau în momentul în care s-a ajuns la sfârșitul fișierului.

În cazul în care ne aflăm la sfârșitul fișierului, funcția `fgets` returnează valoarea logică **FALSE**.

În continuare vom prezenta un exemplu de utilizare a acestei funcții:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fgets
    </TITLE>
 </HEAD>
 <BODY>
   <?PHP
      $fisier = "c:\\test.txt";
      $fd = fopen($fisier,"r");
      $linie = fgets($fd);
      fclose($fd);
      echo "Prima linie din fisier este: ";
      echo $linie;
  ?>
  </BODY>
</HTML>
```

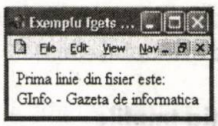


Figura 4: Funcția fgets

În acest exemplu am omis cel de-al doilea parametru. În figura 4 poate fi observat efectul execuției acestui cod.

Funcția fgetss

Această funcție citește o linie dintr-un fișier *HTML*. Spre deosebire de funcția **fgets**, funcția **fgetss** are trei parametri.

Primii doi parametri sunt identici cu cei folosiți în cazul funcției **fgets**. Cel de-al treilea parametru este opțional, este de tip șir de caractere și reprezintă o listă de marcaje *HTML* sau *PHP* pe care funcția nu trebuie să le elimine în momentul citirii.

Această funcție returnează un șir de caractere care reprezintă datele din fișierul *HTML* care au fost citite.

Citirea datelor se încheie în momentul în care s-au citit atâtea caractere câte indică cel de-al doilea parametru sau în momentul în care am ajuns la sfârșitul fișierului.

În continuare prezentăm un exemplu de utilizare a acestei funcții și în figura 6 poate fi observat efectul execuției acestui cod pentru fișierul din figura 5.

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fgetss
    </TITLE>
 </HEAD>
```

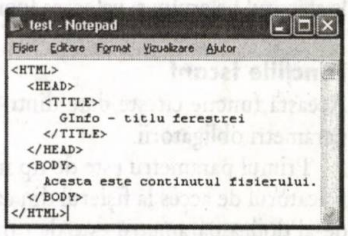


Figura 5: Fișier HTML

```
<BODY>
  <?PHP
    $fis = "c:\\test.html";
    $fd = fopen($fis,"r");
    echo "Continutul fisierului HMTL este:";
    while (!$l = fgetss($fd,filesize($fis))){
      echo $l."<BR>";
    }
  fclose($fd);
?>
 </BODY>
</HTML>
```

Figura 6: Funcția fgetss

Se observă că rezultatul obținut după citirea tuturor liniilor unui fișier *HTML* conține foarte multe rânduri goale. Acest fapt se datorează liniilor care conțin doar marcaje *HTML* și, prin eliminarea lor, șirul returnat devine vid.

Funcția fgetcsv

Această funcție citește date dintr-un fișier *CSV* (*Comma Separated Values - valori separate prin virgulă*).

Un fișier de tip *CSV* are formatul text și conține mai multe înregistrări. Este o reprezentare textuală a unui fișier cu tip. Pe fiecare linie a unui astfel de fișier se află mai multe valori ale unor câmpuri care sunt delimitate printr-un separator. De obicei acest separator este caracterul "," (virgula), de unde și numele acestui tip de fișier.

Funcția `fgetcsv` are trei parametri dintre care primii doi sunt obligatorii. Primul parametru este de tip întreg și reprezintă identificatorul de acces la fișierul de intrare, al doilea parametru este de tip întreg și reprezintă numărul maxim de octeți care se vor procesa la o citire și trebuie să aibă o valoare mai mare decât lungimea celei mai mari linii din fișierul *CSV*, iar al treilea parametru este de tip șir de caractere care trebuie să aibă lungimea 1 și reprezintă separatorul care se va folosi la procesarea datelor.

Funcția returnează, în cazul în care nu ne aflăm la sfârșitul fișierului de intrare, un tablou de șiruri de caractere care reprezintă valorile citite.

În cazul în care ne aflăm la sfârșitul fișierului, funcția `fgetcsv` returnează valoarea logică **`FALSE`**.

În continuare prezentăm un exemplu de utilizare a acestei funcții pentru un fișier *CVS* în care fiecare linie nu are o lungime mai mare de 1000 de caractere:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fgetcsv
  </TITLE>
 </HEAD>
  <?PHP
    $lin = 1;
    $fp = fopen("test.csv","r");
    while ($date = fgetcsv($fp, 1000, ",")) {
      $num = count($date);
      echo "<P> $num câmpuri pe linia
                                  $lin: <BR>";
      $lin++;
      for($c=0; $c < $num; $c++){
        echo $date[$c] . "<br>";
      }
    }
    fclose($fp);
  ?>
 </BODY>
</HTML>
```

Funcția file_get_contents

Această funcție citește conținutul unui fișier de tip text și are doi parametri. Primul parametru este de tip șir de caractere și reprezintă numele fișierului al cărui conținut se va citi, iar cel de-al doilea parametru este opțional și indică dacă pentru găsirea fișierului se va folosi sau nu lista de directoare predefinită.

Funcția returnează un șir de caractere care conține toate datele din fișierul primit ca parametru în cazul în care citirea s-a efectuat cu succes și valoarea logică **`FALSE`** în caz contrar.

Funcția file

Această funcție citește conținutul unui fișier de tip text și are aceeași sintaxă cu funcția `file_get_contents`, dar rezultatul returnat de aceasta este un tablou de șiruri de caractere, fiecare dintre acestea constând într-o linie din fișierul primit ca parametru.

Funcția filetype

Această funcție returnează un șir de caractere care reprezintă tipul unui fișier. Funcția `filetype` are un singur parametru de tip șir de caractere, mai exact numele fișierului pentru care dorim să determinăm tipul. Valorile posibile pentru tipul unui fișier sunt `"pipe"`, `"fifo"`, `"char"`, `"dir"`, `"block"`, `"link"`, `"file"` și `"unknown"`.

Funcția filesize

Această funcție se folosește pentru a afla dimensiunea, exprimată în octeți, a unui fișier. Funcția `filesize` are un singur parametru de tip șir de caractere, mai exact numele fișierului pentru care dorim să determinăm dimensiunea.

Funcția fwrite

Această funcție se folosește pentru a scrie date într-un fișier text sau binar.

Pentru compatibilitate cu toate sistemele de operare, se recomandă ca la deschiderea fișierului, pentru scriere, să se adauge caracterul `"b"` la parametrul care reprezintă modul de deschidere a fișierului.

Funcția `fwrite` are trei parametri. Primul parametru este de tip întreg și reprezintă identificatorul de acces la fi-

șierul în care se dorește să se scrie date, cel de-al doilea parametru este de tip șir de caractere și reprezintă datele care se vor scrie în fișier, iar cel de-al treilea parametru este de tip întreg, opțional și reprezintă numărul de octeți din șirul de caractere care se vor scrie în fișier.

Funcția returnează un număr întreg care reprezintă numărul de octeți pe care a reușit să-i scrie în fișier, în cazul în care nu au apărut erori și valoarea logică **FALSE** în caz contrar.

În continuare prezentăm un exemplu de utilizare a acestei funcții și în figura alăturată poate fi observat efectul execuției codului:

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu fwrite
  </TITLE>
</HEAD>
  <?PHP
      $fis = "c:\\test.txt";
      $sir = "Scriere in fisier";
      $fp = fopen($fis,"a");
      fwrite($fp,$sir);
      echo "Am scris ($sir) in fisierul ($fis).";
      fclose($fp);
  ?>
</BODY>
</HTML>
```

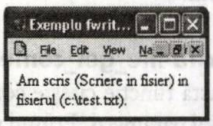


Figura 7: Funcția fwrite

Funcția fputs

Această funcție este o redenumire a funcției `fwrite`, pentru compatibilitate cu limbajul *C/C++*, și are același comportament. În limbajul *C/C++* funcția `fwrite` este folosită pentru a scrie un șir de octeți într-un fișier, spre deosebire de *PHP*, unde șirul de octeți este reprezentat chiar de șirul de caractere.

Funcția fflush

Această funcție forțează salvarea pe disc a conținutului zonelor tampon folosite pentru scrierea într-un fișier.

Funcția `fflush` primește un singur parametru de tip întreg, care reprezintă identificatorul de acces al fișierului pentru care se dorește golirea zonelor tampon.

Funcția returnează valoarea logică **FALSE** în caz de eroare și valoarea logică **TRUE** în caz contrar.

Funcția flock

Această funcție se folosește pentru a bloca sau debloca accesul la un fișier. Necesitatea utilizării funcției `flock` apare în momentul în care mai multe procese doresc să acceseze același fișier și nu se vrea afectarea integrității acestuia.

Funcția are trei parametri. Primul parametru reprezintă identificatorul de acces la conținutul fișierului, cel de-al doilea parametru reprezintă tipul operației de blocare a fișierului, iar al treilea parametru este de tip referință la întreg și va primi valorea logică **TRUE** în cazul în care funcția `flock` va opri execuția *script*-ului până în momentul în care va reuși să blocheze fișierul.

Pentru cel de-al doilea parametru, în limbajul *PHP* sunt disponibile constantele:

- `LOCK_SH` — se folosește în cazul în care fișierul se blochează în mod partajat pentru citirea datelor;
- `LOCK_EX` — se folosește în cazul în care fișierul se blochează în mod exclusiv pentru scrierea datelor;
- `LOCK_UN` — se folosește în cazul în care se dorește deblocarea unui fișier;
- `LOCK_NB` — se folosește împreună cu una dintre primele trei constante dacă nu se dorește ca funcția `flock` să întrerupă execuția *script*-ului până în momentul în care va reuși să efectueze operația cerută; această constantă se adună la una dintre primele trei.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în cazul unui eșec.

Funcția fseek

Această funcție se folosește pentru modificarea poziției de citire sau scriere în interiorul unui fișier.

Funcția `fseek` are trei parametri. Primul parametru este de tip întreg și reprezintă identificatorul de acces la un fișier, al doilea parametru este de tip întreg și indică numărul de poziții cu care se va face modificarea, iar al treilea parametru, tot de tip întreg, este opțional și specifică modul în care se va face modificarea.

Se observă că cel de-al doilea parametru poate avea și valori negative.

Modificarea poziției de citire sau scriere se poate face în trei moduri pentru care sunt definite constante:

- relativ la începutul fișierului — constanta `SEEK_SET`;
- relativ la poziția curentă — constanta `SEEK_CUR`;
- relativ la sfârșitul fișierului — constanta `SEEK_SET`.

În cazul în care cel de-al treilea parametru lipsește se va folosi ca valoare pentru acesta constanta `SEEK_SET`.

Funcția returnează o valoare întreagă egală cu 0, în cazul în care nu au apărut erori, și egală cu -1 în caz contrar.

Această funcție nu poate fi folosită pentru fișiere care sunt accesate folosind unul dintre protocoalele *HTTP* sau *FTP*.

Funcția rewind

Această funcție schimbă poziția de citire sau scriere într-un fișier la începutul acestuia și are un singur parametru care specifică identificatorul de acces la fișier.

Rezultatul returnat de funcția `rewind` este același cu cel returnat de funcția `fseek`.

Apelul acestei funcții este echivalent cu apelul funcției `fseek` cu primul parametru având ca valoare identificatorul de acces al fișierului, al doilea parametru având valoarea `0` și cel de-al treilea parametru având valoarea `SEEK_SET` chiar dacă lipsește.

Funcția ftell

Această funcție se folosește pentru a afla poziția curentă de la care se va face următoarea citire sau scriere într-un fișier, relativ la începutul acestuia.

Funcția `ftell` primește ca parametru identificatorul de acces al fișierului pentru care se aplică și returnează o valoare întreagă reprezentând poziția din fișier, în caz de succes sau valoarea logică **`FALSE`** în caz de eșec.

Funcția ftruncate

Această funcție realizează redimensionarea fișierului cu o valoare primită ca parametru.

Funcția `ftruncate` are doi parametri. Primul dintre aceștia este identificatorul de acces al fișierului care se redimensionează, iar al doilea parametru este de tip întreg și reprezintă noua dimensiune a fișierului.

Trebuie ținut cont de faptul că în cazul în care cel de-al doilea parametru are o valoare mai mare decât dimensiunea curentă a fișierului va fi alocat, pe disc, spațiu suplimentar care nu va fi inițializat.

În cazul în care cel de-al doilea parametru are o valoare mai mică decât dimensiunea curentă a fișierului, atunci toate datele din fișier, care se află după poziția indicată de noua valoare, se vor pierde.

Funcția delete

Această funcție este mai degrabă o redenumire a funcției `unlink`, care se folosește pentru a șterge un fișier de pe disc.

Funcția `delete` a fost inclusă în pachetul de funcții oferit de limbajul *PHP* pentru cei care au programat în limbaje precum *Pascal* și nu au avut la dispoziție funcția `unlink`.

Funcția `unlink` primește ca parametru un nume de fișier și returnează valoarea logică **`TRUE`** în cazul în care ștergerea s-a efectuat cu succes.

În cazul în care ștergerea nu s-a putut efectua, funcția returnează valoarea logică **`FALSE`**.

Manipularea directoarelor

În cele ce urmează vom prezenta cele mai importante funcții și clase oferite de limbajul *PHP* pentru manipularea directoarelor.

Funcția getcwd

Această funcție returnează un șir de caractere care reprezintă directorul curent.

Funcția `getcwd` nu are nici un parametru.

În continuare prezentăm un exemplu de utilizare al acestei funcții al cărei rezultat poate fi observat în figura alăturată:

```
<HTML>
 <HEAD>
  <TITLE>
     Exemplu getcwd
  </TITLE>
 </HEAD>
 <BODY>
    <?PHP
      $cdir = getcwd();
      echo "Directorul curent este: ".$cdir;
  ?>
 </BODY>
</HTML>
```

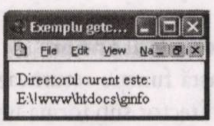


Figura 8: Funcția getcwd

Funcția chdir

Această funcție se folosește pentru a schimba directorul curent și primește ca parametru un șir de caractere care reprezintă noul director.

Funcția `chdir` returnează valoarea logică **`TRUE`** atunci când schimbarea directorului curent s-a efectuat cu succes și valoarea logică **`FALSE`** în caz contrar.

Funcția chroot

Această funcție se folosește pentru a schimba directorul rădăcină pentru procesul curent și primește ca parametru un șir de caractere care reprezintă noul director rădăcină.

Funcția `chroot` returnează valoarea logică **`TRUE`** atunci când schimbarea directorului rădăcină s-a efectuat cu succes și valoarea logică **`FALSE`** în caz contrar.

Această funcție nu este implementată pe platformele de operare *Microsoft Windows* și nu este recomandată folosirea ei decât în cazul în care interpretorul *PHP* este utilizat folosind metoda de execuție *CGI*.

Funcția opendir

Această funcție se folosește pentru a deschide un director în scopul citirii conținutului său.

Funcția `opendir` are un singur parametru de tip șir de caractere care reprezintă numele directorului al cărui conținut se va prelucra.

Această funcție returnează un identificator de acces la conținutul directorului și este de tipul resursă.

În cazul în care nu se reușește deshiderea directorului, funcția returnează valoarea logică **`FALSE`**.

Funcția closedir

Această funcție se folosește pentru a închide un director care a fost deschis pentru citire folosind funcția `opendir`.

Funcția `closedir` are un singur parametru de tipul resursă, care reprezintă identificatorul de acces al directorului care se dorește a fi închis.

Această funcție nu returnează nici o valoarea, ea având tipul **`void`**.

Funcția readdir

Funcția `readdir` se folosește pentru a citi următoarea intrare disponibilă într-un director.

Ea are un singur parametru de tip resursă, care reprezintă identificatorul de acces al directorului din care se va citi.

Funcția `readdir` returnează un șir de caractere reprezentând numele următoarei intrări din director, în cazul în care nu am ajuns la sfârșitul acestuia sau nu au apărut erori

și valoarea logică **FALSE** în caz contrar.

În continuare vă prezentăm un exemplu de utilizare a funcțiilor `opendir`, `readdir` și `closedir`, iar în figura alăturată poate fi observat efectul execuției codului următor:

Figura 9: Funcțiile opendir, readdir, closedir

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu opendir,
              readdir,closedir
  </TITLE>
 </HEAD>
   <?PHP
     $dir = "E:\\";
     $dh = opendir($dir);
     echo "Continutul directorului $dir<BR>";
     while (false !== ($fis = readdir($dh))){
       echo "$fis<BR>";
     }
     closedir($dh);
  ?>
 </BODY>
</HTML>
```

Funcția rewinddir

Această funcție resetează poziția curentă dintr-un director, și anume, dacă după un anumit număr de citiri este apelată această funcție și după aceeea se mai citește o intrare din directorul respectiv, atunci se va returna prima intrare din director.

Funcția are un singur parametru de tipul resursă, care reprezintă identificatorul de acces al directorului din care se va citi și nu returnează nici o valoare.

Clasa dir

Această clasă încapsulează funcțiile `opendir`, `closedir`, `readdir` și `rewinddir`. Structura acesteia este următoarea:

```
class dir{
  dir(string director);
  string path;
  resource handle;

  string read();
  void rewind();
  void close();
}
```

Constructorul clasei `dir` primește ca parametru un șir de caractere care reprezintă numele directorului care se va citi și apelează funcția `opendir`.

Funcțiile `read`, `rewind` și `close` din cadrul acestei clase au același comportament cu funcțiile `readdir`, `rewinddir`, respectiv `closedir` și se deosebesc de acestea prin simplul fapt că nu mai au parametri, deoarece identificatorul de acces la director este de asemenea încapsulat în cadrul clasei `dir`.

Figura 10: Clasa dir

În continuare vă vom prezenta un exemplu de utilizare a acestei clase care reprezintă implementarea cu clase a exemplului anterior și rezultatul execuției codului poate fi observat în figura 10.

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu dir
  </TITLE>
 </HEAD>
   <?PHP
     $d = dir("E:\\");
     echo "Continutul directorului $d->path
                                  <BR>";
     while (false !== ($fis = $d->read())){
       echo "$fis<BR>";
     }
     $d->close();
  ?>
 </BODY>
</HTML>
```

Funcția mkdir

Această funcție se folosește pentru a crea un director nou și primește ca parametru un șir de caractere care reprezintă numele directorului care se va crea și atributele acestuia (parametru de tip întreg) în format **nix*.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în caz de eșec.

Funcția rmdir

Această funcție se folosește pentru a șterge un director de pe disc și primește ca parametru un șir de caractere care reprezintă numele directorului care va fi șters.

Directorul care se dorește a fi șters trebuie să nu conțină nici un fișier sau subdirector.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în caz de eșec.

Alte funcții

În cele ce urmează vom prezenta un set de funcții care facilitează lucrul cu fișiere și/sau directoare .

Funcția basename

Această funcție returnează numele de bază al unui fișier sau director sub forma unui șir de caractere.

Funcția are doi parametri de tip șir de caractere. Primul parametru reprezintă o cale către un fișier sau director, iar al doilea este opțional și reprezintă sufixul fișierului.

În continuare vă prezentăm un exemplu de utilizare a acestei funcții și în figura alăturată se poate observa efectul execuției codului următor:

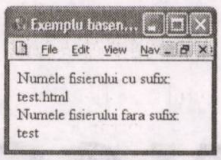


Figura 11:
Funcția basename

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu basename
   </TITLE>
 </HEAD>
   <?PHP
      $cale = "c:\\test.html";
      echo "Numele fisierului cu sufix:<BR>";
      echo basename($cale)."<BR>";
      echo "Numele fisierului fara sufix:<BR>";
      echo basename($cale,".html");
  ?>
 </BODY>
</HTML>
```

Funcția dirname

Această funcție returnează numele directorului părinte al unui fișier sau director sub forma unui șir de caractere.

Funcția are un singur parametru de tip șir de caractere. care reprezintă o cale către un fișier sau director.

În continuare vă prezentăm un exemplu de utilizare a acestei funcții și în figura alăturată se poate observa efectul execuției codului următor:

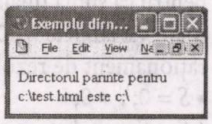


Figura 12:
Funcția dirname

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu dirname
   </TITLE>
 </HEAD>
   <?PHP
      $cale = "c:\\test.html";
      echo "Directorul parinte pentru ";
      echo $cale." este ".dirname($cale);
  ?>
 </BODY>
</HTML>
```

Funcția pathinfo

Această funcție se folosește pentru a afla informații referitoare la o cale către un fișier sau director.

Funcția `pathinfo` primește ca parametru un șir de caractere care reprezintă calea către un fișier sau director.

Funcția returnează un tablou ale cărui valori sunt de tipul șir de caractere și care conține elementele `"dirname"`, `"basename"` și `"extension"`.

Elementele `"dirname"` și `"basename"` au valori identice cu cele returnate de funțiile cu același nume, iar elementul `"extension"` conține extensia fișierului.

În continuare vă prezentăm un exemplu de utilizare a acestei funcții și în figura 13 se poate observa efectul execuției codului următor:

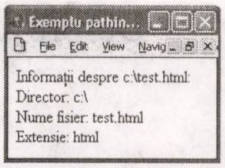


Figura 13:
Funcția pathinfo

```
<HTML>
 <HEAD>
  <TITLE>
   Exemplu pathinfo
   </TITLE>
 </HEAD>
   <?PHP
      $cale = "c:\\test.html";
      $info = pathinfo($cale);
      echo "Informații despre $cale:<BR>";
      echo "Director: ";
      echo $info["dirname"]."<BR>";
      echo "Nume fisier: ";
      echo $info["basename"]."<BR>";
      echo "Extensie: ";
      echo $info["extension"]."<BR>";
  ?>
 </BODY>
</HTML>
```

Funcția file_exists

Această funcție se folosește pentru a determina dacă un fișier există sau nu pe disc.

Funcția `file_exists` are un singur parametru de tip șir de caractere care reprezintă numele fișierului pentru care se verifică existența fizică pe disc.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în caz de eșec.

Această funcție nu poate fi folosită pentru fișiere aflate la distanță decât dacă sunt partajate de pe o platformă *Microsoft Windows* pe care a fost configurată componenta *File and Print Sharing* sau dacă sunt partajate pe o platformă **nix* pe care a fost instalat pachetul *Samba*.

Funcția copy

Această funcție se folosește pentru a copia un fișier și are doi parametri de tip șir de caractere. Primul dintre aceștia reprezintă numele fișierului sursă și cel de-al doilea reprezintă numele fișierului destinație.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în caz de eșec.

Funcția rename

Această funcție se folosește pentru a redenumi un fișier și are doi parametri de tip șir de caractere. Primul parametru reprezintă numele fișierului și cel de-al doilea reprezintă noul numele al acestuia.

Funcția returnează valoarea logică **TRUE** în caz de succes și valoarea logică **FALSE** în caz de eșec.

Va urma...

În articolul următor vom prezenta în detaliu câteva facilități importante oferite de limbajul *PHP*, dintre care amintim interoperabilitatea acestui limbaj cu platforma *JAVA* și crearea dinamică a imaginilor pentru publicarea pe diferite *site*-uri.

Conceptul de ALGORITM - abordare MODERNĂ

Marin Vlada

În cadrul acestui ultim articol, dedicat unei abordări moderne a conceptului de algoritm, vom prezenta șase probleme simple care ilustrează modul de elaborare al algoritmilor de rezolvare pentru acestea.

Problema 1: Sumă de valori

Se consideră vectorul $X = (x_1, x_2, ..., x_n)$ ale cărui componente $x_1, x_2, ..., x_n$ sunt numere reale. Să se calculeze suma S a componentelor vectorului X, adică:

$$S = \sum_{i=1}^{n} x_i.$$

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt:

- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, ..., x_n)$ - vectorul cu componente reale.

Date de ieșire

Datele de ieșire (*De*) pentru această problemă sunt constituite de suma:

$$S = \sum_{i=1}^{n} x_i.$$

Rezolvare

Rezolvarea problemei constă în câțiva pași care sunt prezentați în continuare.

a) *Raționamentul de rezolvare* (*R*)

Pentru a rezolva această problemă vom defini următorul șir de valori:

- $S_0 = 0$;
- $S_i = S_{i-1} + x_i$, $1 \le i \le n$.

Prin inducție matematică se poate demonstra foarte ușor că $S_n = \sum_{i=1}^{n} x_i$. Prin urmare, este necesar să se elaboreze procesul de calcul P care să calculeze succesiv valorile S_0, S_1, ..., S_n, valori care generează suma $S = S_n$.

Datorită faptului că valorile intermediare S_{i-1}, $1 \le i \le n$ nu vor mai fi folosite în alte scopuri în afară de calcularea valorilor S_i, aceste variabile se pot înlocui cu una singură și pentru că suma finală S este egală cu S_n, atunci vom înlocui toate variabilele S_i, $0 \le i \le n$, cu S și vom avea următorul raționament de rezolvare:

- $S = 0$, $i = 0$;
- $S = S + x_i$, $1 \le i \le n$.

b) *Mulțimea variabilelor* (*V*)

Variabilele definite de raționamentul R, care vor fi utilizate în procesul de calcul P, sunt:

- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, ..., x_n)$ - vectorul cu componente reale;
- S - variabila care stochează sumele parțiale și suma finală.

c) *Procesul de calcul* (*P*) *și reprezentarea algoritmului*

După *citirea* (`read`) unei valori pentru variabila n și *citirea* celor n valori care reprezintă componentele vectorului X, toate aceste valori vor fi stocate în memoria internă (prin intermediul mediului de intrare *Mi*), în locații de memorie la care se poate avea acces prin intermediul instrucțiunilor (executate de o mașină virtuală).

Conform relației de recurență prezentată anterior, inițial trebuie ca în locația de memorie asociată lui S să fie stocată valoarea 0. Evident, pentru calculul valorilor intermediare se va utiliza intrucțiunea repetitivă cu contor `for`.

După încheierea execuției acestei instrucțiuni, în locația de memorie asociată variabilei S se va afla valoarea cerută de problemă (conform raționamentului de rezolvare), valoare care va fi citită din memorie prin intermediul mediului de ieșire *Me* și afișată pe ecran.

Prin urmare, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm Suma_valori;
  Integer n;
  Real s, x(500);
begin
   read n;
   read x1, x2, ..., xn;
   S ← 0;                          // starea inițială
   for i = 1, n do
      S ← S + xi;
   write S;
end
```

Problema 2: Configurația electronică

Dacă Z este numărul atomic al unui element chimic, să se elaboreze un algoritm pentru completarea straturilor energetice din învelișul electronic al atomului. Se presupune că straturile se completează în ordine cu numărul maxim de electroni $2 \cdot i^2$, unde i este numărul stratului, excepție ultimul strat care se completează cu restul de electroni. Să se determine numărul straturilor care se completează și pentru fiecare strat, numărul electronilor care se află pe acesta.

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt date de numărul atomic Z al unui element chimic.

Date de ieșire

Datele de ieșire (*De*) sunt constituite din:

- n - numărul de straturi completate;
- $S = (s_1, s_2, ..., s_n)$ - lista care va conține numărul de electroni pentru fiecare strat completat.

Rezolvare

a) *Raționamentul de rezolvare* (*R*)

Trebuie conceput un proces de calcul care să contorizeze numărul straturilor care s-au completat la un moment dat și să calculeze numărul total de electroni (sume parțiale) repartizați pe straturi până la un moment dat.

Conform enunțului, stratul i, $i > 0$, se completează cu $s_i = 2 \cdot i^2$ electroni. Vom nota cu $Sp = s_1 + s_2 + ... + s_i$ suma parțială (numărul electronilor repartizați) la momentul i.

La fiecare pas, procesul de calcul trebuie să realizeze:

- incrementarea contorului pentru stratul completat;
- completarea stratului i cu $s_i = 2 \cdot i^2$ electroni;
- calculul sumei parțiale $Sp = s_1 + s_2 + ... + s_i$;
- repetarea operațiilor precedente *cât timp* $Sp < Z$.

Pentru ca raționamentul să fie corect, în momentul în care nu mai este satisfacută condiția $Sp < Z$, se va realiza completarea ultimului strat cu numărul de electroni dați de diferența dintre numărul atomic Z și suma Sp, după ce din suma Sp se scade numărul de electroni adăugați ultima dată.

b) *Mulțimea variabilelor* (*V*)

Variabilele definite de raționamentul *R*, care vor fi utilizate în procesul de calcul *P*, sunt:

- *Z* - numărul atomic al elementului chimic;
- *Sp* - variabila care va conține suma parțială;
- *n* - numărul de strate completate;
- *S* - vectorul care va conține numărul electronilor pentru fiecare strat.

c) *Procesul de calcul* (*P*) *și reprezentarea algoritmului*

Conform raționamentului de rezolvare prezentat anterior, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm Configuratia_electronica;
  Integer Z,n,Sp,s(7);
begin
   read Z;  // nr. atomic pentru elementul chimic
   // inițializări
   n ← 0;  // contor pentru nr. stratului completat
   Sp ← 0;                  // suma parțială a electronilor
   while (Sp < Z) do begin
         n ← n + 1;                        // incrementare
         sn ← 2 · n · n;                   // completare strat
         Sp ← Sp + sn;                     // suma parțială
   end;
   Sp ← Sp - sn;
   sn ← Z - Sp;    // completarea ultimului strat
   write "Nr. de straturi completate n =", n;
   write "Nr. electronilor pe straturi este:";
   write s1, s2, ..., sn;
end
```

Problema 3: Algoritmul lui Euclid

Dându-se două numere întregi a și b, se cere să se calculeze cel mai mare divizor comun al lor.

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt constituite din cele două numere întregi a și b.

Date de ieșire

Datele de ieșire (*De*) sunt date de cel mai mare divizor comun al celor două numere.

Rezolvare

a) *Raționamentul de rezolvare* (*R*)

Pentru a rezolva această problemă trebuie ținut cont de teorema împărțirii cu rest.

Considerăm următorul șir de împărțiri succesive:

$a = b \cdot c_1 + r_1, b > r_1$

$b = r_1 \cdot c_2 + r_2, r_1 > r_2$

$r_1 = r_2 \cdot c_3 + r_3, r_2 > r_3$

$r_2 = r_3 \cdot c_4 + r_4, r_3 > r_4$

...

$r_{n-1} = r_n \cdot c_{n+1} + r_{n+1}, r_n > r_{n+1}$

Se poate observa că șirul resturilor $(r_i)_{i>0}$ este *mărginit*, *descrescător* și *convergent* către 0. Pentru cazul în care r_{n+1} este egal cu 0, *Euclid* a demonstrat că cel mai mare divizor comun al numerelor *a* și *b* este r_n, adică *ultimul rest nenul* din șirul de împărțiri succesive.

În concluzie, trebuie realizat un proces de calcul (și, implicit un algoritm) pentru a executa operațiile din șirul de împărțiri succesive. Procesul se va termina în momentul în care restul unei împărțiri din șirul de mai sus este 0.

Șirul împărțirilor de mai sus este corect doar în cazul în care *a* și *b* au același semn (amândouă numerele sunt pozitive sau negative în același timp). În caz contrar, trebuie să considerăm valoarea absolută a acestora, rezultatul final fiind același.

b) *Mulțimea variabilelor* (*V*)
Deoarece resturile împărțirilor intermediare (de la prima până la penultima) nu mai sunt folosite în alte scopuri, mulțimea variabilelor utilizate în procesul de calcul este formată doar din variabilele *a*, *b* și o variabilă auxiliară *r*, care, la fiecare pas, va conține restul împărțirii lui *a* la *b*.

c) *Procesul de calcul* (*P*) *și reprezentarea algoritmului*
Ținând seama că nu este necesară stocarea valorilor intermediare ale resturilor, rolul variabilelor *a* și *b* se va schimba la executarea fiecărei împărțiri din șirul împărțirilor succesive. Procesul de împărțire se repetă *până când r* devine 0. Se poate observa că valoarea câtului nu intervine în dinamica procesului de calcul.

În continuare prezentăm algoritmul de rezolvare a problemei:

```
Algorithm EUCLID_1;
  Integer a,b,r;
begin
  read a,b;
  repeat
    r ← a mod b;
    a ← b;      // împărțitorul devine deîmpărțit
    b ← r;            // restul devine împărțitor
  until r = 0;
  write "Cel mai mare divizor comun: ", a;
end
```

Folosind doar două variabile *a* și *b* (care, inițial vor stoca valorile datelor de intrare), se poate concepe un alt proces de calcul prin înlocuirea împărțirilor succesive cu scăderi repetate (operația de împărțire fiind, de fapt, un șir de scăderi repetate), obținându-se algoritmul lui *Nicomachus*.

```
Algorithm EUCLID_2;
  Integer a,b;
begin
  read a,b;
  repeat
    if a > b then
      a ← a - b;
    else
      b ← b - a;
  until a = b;
  write "c.m.m.d.c.=", a;
end
```

Problema 4: Algoritm de sortare

Dacă $x_1, x_2, \ldots, x_n$ sunt componentele reale ale vectorului $X = (x_1, x_2, \ldots, x_n)$, să se găsească o permutare σ a componentelor vectorului astfel încât acestea să fie sortate crescător, adică:
$X = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ și $x_{\sigma(1)} \le x_{\sigma(2)} \le \ldots \le x_{\sigma(n)}$.

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt:
- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, \ldots, x_n)$ - vectorul cu componente reale.

Date de ieșire

Datele de ieșire (*De*) sunt date de vectorul:
$X = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ și $x_{\sigma(1)} \le x_{\sigma(2)} \le \ldots \le x_{\sigma(n)}$.

Rezolvare

a) *Raționamentul de rezolvare* (*R*)
Există foarte mulți algoritmi de sortare, unii foarte performanți, alții mai puțin performanți [23]. Vom aplica un algoritm foarte utilizat, bazat pe metoda interschimbărilor, cunoscut sub numele de *metoda bulelor*. Metoda constă într-un proces de calcul principal în cadrul căruia, la fiecare pas, se parcurg componentele vectorului X (numărul de pași ai procesului fiind nedefinit, terminarea procesului fiind dată de obținerea sortării dorite). Prin parcurgerea componentelor vectorului X se înțelege, în acest context, un subproces de calcul care se execută în $(n - 1)$ pași și constă în analiza perechilor (x_i, x_{i+1}), $1 \le i \le n$, ceea ce înseamnă compararea valorilor celor două componente și efectuarea sau nu a interschimbării acestora în functie de tipul sortării (crescătoare sau descrescătoare).

Efectuarea interschimbării componentelor x_i, x_{i+1} se poate monitoriza prin utilizarea unei variabile intermediare (de lucru), notată, de exemplu, prin K și care poate să aibă valorile 0, dacă nu se realizează interschimbarea, sau 1, dacă se realizează interschimbarea. Această variabilă cu rol de indicator (semafor) se va utiliza pentru a controla terminarea procesului de calcul principal, și anume pentru a încheia parcurgerea componentelor vectorului X. Pentru acest scop, înainte de fiecare parcurgere, valoarea lui K se setează cu 0. În cazul în care în timpul parcurgerii componentelor vectorului (executării subprocesului de calcul pentru analiza celor $n - 1$ perechi pentru care se va utiliza instrucțiunea **for**) se va efectua cel puțin o interschimbare, indicatorul K se va seta cu valoarea 1. În cazul în care nu s-a efectuat nici o interschimbare, valoarea lui K rămâne neschimbată, adică 0, ceea ce este echivalent cu faptul că

valorile vectorului X sunt sortate. Prin urmare, la sfârșitul fiecărei parcurgeri se va verifica dacă valoarea indicatorului K este 0, caz în care procesul de calcul principal se va termina (se va utiliza instrucțiunea **repeat**). Evident, metoda utilizează structura vectorului X ca suport de stocare a componentelor sortate. Din acest motiv, pentru ca operația de interschimbare să se realizeze corect, se va folosi o variabilă intermediară (de lucru) notată *save*, variabilă care va stoca temporar valoarea x_i și care, ulterior, se va stoca în locul valorii x_{i+1}. Secvența corectă de instrucțiuni (*metoda celor trei scaune*) este următoarea (execuția secvențială a instrucțiunilor):

```
save ← x(i);
x(i) ← x(i+1);
x(i+1) ← save;
```

Operația de interschimbare prin simplele operații de atribuire `x(i) ← x(i+1)` și `x(i+1) ← x(i);` este incorectă, deoarece s-ar pierde valoarea `x(i)` și în final vectorul X ar conține alte valori și nu cele inițiale (citite).

În concluzie, procesul de calcul principal și subprocesul de parcurgere a componentelor vectorului X, se poate reprezenta schematic astfel:

Parcurgerea 1:
  $k \leftarrow 0$
  Analiza perechilor: (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n).
  Verificare condiție de oprire: k=0?
Parcurgerea 2:
  $k \leftarrow 0$
  Analiza perechilor: (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n).
  Verificare condiție de oprire: k=0?
  ...
Parcurgerea ?:
  $k \leftarrow 0$
  Analiza perechilor: (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n).
  Verificare condiție de oprire: k=0.

b) *Mulțimea variabilelor* (V)
Variabilele definite de raționamentul R, care vor fi utilizate în procesul de calcul P, sunt:
- X - vector cu componente reale;
- n - numărul de componente ale vectorului X;
- K - indicator pentru interschimbare;
- *save* - variabila folosită la interschimbări.

c) *Procesul de calcul* (P) *și reprezentarea algoritmului*
Conform raționamentului de rezolvare prezentat anterior, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm sortare_crescatoare;
  Integer n,k;
  Real save,X(500);
begin
 read n;
 read x1, x2, …, xn;
  // procesul de calcul principal
  repeat
    k ← 0;                          // inițializare
   // parcurgerea vectorului - subprocesul de calcul
    for i = 1,n-1 do begin
      if xi > xi+1 then begin // interschimbare
        k ← 1;                      // setare
        save ← xi;
        xi ← xi+1;
        xi+1 ← save;
      end
    end
  until k=0;
  write "Vectorul sortat: ", x1, x2, …, xn;
end
```

Problema 5: Puterile lui 2

Fie k un număr natural. Să se determine numărul cifrelor și cifrele numărului 2^k.

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt constituite din numărul k.

Date de ieșire

Datele de ieșire (*De*) pentru această problemă sunt:
- n - numărul cifrelor lui 2^k;
- $X = (x_1, x_2, ..., x_n)$ - vectorul cifrelor lui 2^k.

Rezolvare

a) *Raționamentul de rezolvare* (R)
Evident, problema ar fi fără sens dacă s-ar rezolva printr-o singură instrucțiune dintr-un limbaj de programare. Acest lucru se poate realiza doar dacă ar exista restricția $k < 31$. Ținând seama de reprezentarea tipului *integer* în memoria internă a calculatorului, majoritatea microprocesoarelor și limbajelor de programare pot stoca/reprezenta o valoare întreagă doar pe 4 bytes (32 biți).

Prin urmare, $2^{31} - 1 = 2147483647$ este cea mai mare valoare întreagă pe care o putem stoca. Este necesar să concepem un algoritm pentru calculul puterilor 2^k, $k > 30$. Vom pleca de la următorul tabel (generat cu ajutorul unui program simplu):

k	1	2	3	4	5	6	7	8	9	10	11	12
2^k	2	4	8	16	32	64	128	256	512	1024	2048	4096

Pentru a putea depăși obstacolul de mai sus, deci pentru a putea calcula valoarea 2^k pentru valori ale lui k mai mari decât 30, trebuie utilizat un vector cu elemente de tip întreg care să memoreze cifrele valorii care trebuie calculate (de fapt, acest lucru s-a cerut și în enunț în cadrul datelor de ieșire). Această decizie implică faptul că trebuie să se elaboreze un algoritm care să execute operațiile care rezultă din înmulțirea unui număr natural cu 2.

Prin urmare, dacă vom considera acest vector $X = (x_1, x_2, ..., x_n)$ - vectorul cifrelor lui 2^k, procesul de calcul prin-

cipal trebuie să aibă (k - 1) pași, și anume conform tabelului următor:

$$\begin{array}{cccccccc} k & 3 & 4 & \dots & i & \dots & k \\ 2^k & 4 & 8 & \dots & x^i_n \dots x^i_2 x^i_1 & \dots & x^k_n \dots x^k_2 x^k_1 \end{array}$$

La fiecare pas $i \in \{2, 3, \dots, k - 1\}$ trebuie elaborat subalgoritmul (un subproces de calcul) care să realizeze operațiile pentru obținerea cifrelor valorii 2^{i+1} cu ajutorul cifrelor (memorate în componentele vectorului X) valorii $2^i = x^i_n \dots x^i_2 x^i_1$. Conform algoritmului elementar de înmulțire cu 2 a unui număr natural apare o cifră de transport. Vom nota cu c variabila care memorează la fiecare pas $j \in \{1, 2, \dots, n\}$ cifra de transport. Indicele i superior atașat fiecărui element din vectorul X semnifică faptul că în acesta se află valorile cifrelor calculate la pasul i. Cifrele valorii 2^{i+1}, respectiv valoarea de transport la fiecare pas, se vor calcula astfel:

```
x_j ← (x_j · 2 + c) mod 10;
c ← (x_j · 2 + c) div 10;        // cifra de transport
```

Numărul n de cifre ale valorii 2^k nu este cunoscut și se va modifica la fiecare pas. La început $n = 1$ și în continuare, la fiecare pas i, în cazul în care la sfârșitul subprocesului de calcul, care înmulțește pe 2^i cu 2, valoarea cifrei de transport este 1, atunci n se va incrementa cu 1 și valoarea lui x_n va fi 1. Deci, variabila n se va folosi pentru contorizarea cifrelor valorii 2^k.

Observație: Chiar dacă rationamentul prezentat este corect, în cazul elaborării unui algoritm și unui program pentru rezolvarea problemei, prin testarea și verificarea programului se vor obține rezultate greșite. Rezultatele incorecte provin de la simplul motiv că cifra de transport nu este calculată corect, și anume valoarea sa este calculată folosind cifra valorii 2^{i+1} și a valorii 2^i (avem în vedere execuția secvențială a instrucțiunilor și dinamica valorilor locațiilor din memoria internă). Este nevoie de utilizarea unei variabile auxiliare s pentru a reține la fiecare pas j al subprocesului considerat valoarea lui x_j și astfel vom avea:

```
s ← x_j;                         // variabila auxiliară
x_j ← (x_j · 2 + c) mod 10;
c ← (s · 2 + c) div 10;          // cifra de transport
```

b) *Mulțimea variabilelor* (V)

Variabilele definite de raționamentul R, care vor fi utilizate în procesul de calcul P, sunt:

- k - puterea pentru valoarea care trebuie calculată;
- $X = (x_1, x_2, \dots, x_n)$ - vectorul cifrelor lui 2^k;
- n - numărul de cifre ale lui 2^k;
- c - variabila care stochează valoarea cifrei de transport;
- s - variabila auxiliară folosită pentru calculul cifrelor lui 2^k.

c) *Procesul de calcul* (P) *și reprezentarea algoritmului*

În continuare prezentăm algoritmul de rezolvare (procesul de calcul) a problemei:

```
Algorithm Puterile_lui_2;
  Integer n,k,c,s,X(2000);
begin
  write "Program pentru calcularea valorii 2^k";
  write "Introduceți puterea k:";
  read k;
  // inițializări
  n ← 1;
  x_n ← 2;
  for i = 2, k do begin
    // parcurgerea vectorului X
    c ← 0;                          // inițializare
    for j = 1, n do begin
      s ← x_j;                      // salvare
      x_j ← (x_j · 2 + c) mod 10;   // cifra puterii
      c ← (s · 2 + c) div 10;       // cifra de transport
    end
    if c = 1 then begin
      n ← n + 1;
      x_n ← 1;
    end
  end
  // scrierea cifrelor
  write "Numărul are", n, "cifre.";
  write "Numărul este:";
  for i = n, 1, -1 do
    write Xi;
end
```

Folosind un astfel de algoritm, se poate elabora un program corect pentru problema în studiu. De exemplu, dând valorile 1000 și 2000 lui k, rezultatele obținute au 302 cifre, respectiv 603 cifre.

În figura următoare se poate observa rezultatul furnizat de un program corect pentru valorile 1000 și 2000 date lui k, iar 2^{1000} și 2^{2000} au aceleași ultime 4 cifre, egale cu 9376.

```
Program pentru calcularea valorii 2^k
Introduceti puterea k: 1000

Numarul are 302 cifre.
Numarul este:
1071508607186267320948425049060001810561
4048117055336074437503883703510511249361
2249319837881569585812759467291755314682
5187145285692314043598457757469857480393
4567774824230985421074605062371141877954
1821530464749835819412673987675591655439
4607706291457119647768654216766042983165
2624386837205668069376
```

a) $k = 1000$

```
Program pentru calcularea valorii 2^k
Introduceti puterea k: 2000

Numarul are 603 cifre.
Numarul este:
1148130695274254524232833201177681984022
3177020886952004776427368257662613923703
1385665948631650626991844596463898746277
3447118960863055331425931356166653185391
2998914531228000068877914824004487142892
6990063486244781615463646388363947317026
0404663539709049965581623988089446296056
2331164953616422197033268134416890898445
8505602379484807914058900934776500429002
7167066258305220081322362812917612678833
1720659899539641812702177985840404215985
3183251540889433902091920554957783589672
0391600819572166305827553804255837260155
2834878641943205450891527578388262517543
5528800822842770817965453762184851149029
376
```

b) $k = 2000$

Figura 1

Problema 6: Suma puterilor unei matrice

Dacă $A = \|a_{ij}\|_{i,j=1,n}$ este o matrice pătratică de ordin n cu elemente reale, să se calculeze matricea $X = A + A^2 + ... + A^k$ pentru $k > 1$.

Date de intrare

Datele de intrare (*Di*) pentru această problemă sunt:

- k - puterea;
- n - ordinul matricei A;
- $A = \|a_{ij}\|_{i,j=1,n}$ - matricea.

Date de ieșire

Datele de ieșire (*De*) pentru această problemă sunt constituite de matricea $X = \|x_{ij}\|_{i,j=1,n}$, $X = A + A^2 + ... + A^k$.

Rezolvare

a) *Raționamentul de rezolvare* (*R*)

Procesul de calcul principal trebuie să calculeze suma puterilor $A, A^2, ..., A^k$. Acest proces de calcul va avea $(k - 1)$ pași. Inițial, procesul de calcul trebuie să aibă loc după atribuirea $X \leftarrow A$. La pasul $i \in \{2, 3, ..., k\}$ trebuie să se calculeze A^i (este necesară definirea unei proceduri pentru determinarea produsului a două matrice pătratice) și să se execute instrucțiunea $X \leftarrow X + A^i$.

Ținând seama că algoritmul este folosit ca "sursă" pentru codificarea într-un limbaj de programare, calculul lui A^i trebuie realizat de o procedură. Este necesară folosirea unei variabile care să salveze valoarea lui A^i și care va fi utilizată de procedura de determinare a valorii $A^{i+1} = A \cdot A^i$. Dacă această variabilă este S, inițial, înaintea procesului de calcul principal, trebuie ca S să fie inițializat cu A.

Dacă presupunem că este definită procedura `produs (n, A, B, C)` pentru calculul produsului `C = A · B`, unde n este ordinul matricelor A, B și C, atunci structura generală a procesului de calcul principal este următoarea (matricea P se va utiliza pentru returnarea produsului $A \cdot S$):

```
begin
 read n,k;
 read A = ||a_ij||_i,j=1,n;
  // inițializări
 X ← A;
  S ← A;
  // procesul de calcul principal
 for i = 2, k do begin
  produs(n, A, S, P);
    X ← X + P;
    S ← P;                          // salvare
 end
 write X = ||x_ij||_i,j=1,n;
end
```

Observație: Există limbaje de programare care oferă operații (atribuirea și suma) cu tablouri.

Pentru elaborarea procedurii `produs(n, A, B, C)`, vom porni de la faptul că elementele matricei C se calculează după regula (linia i a lui A se "înmulțește" ca produs scalar cu coloana j a lui B): $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$, unde $i, j \in \{1, 2, ..., n\}$.

Pentru determinarea elementelor matricei C se vor utiliza trei instrucțiuni **for** imbricate (un proces de calcul principal care să parcurgă liniile $i \in \{1, 2, ..., n\}$, un subproces de calcul care să parcurgă coloanele $j \in \{1, 2, ..., n\}$ și pentru fiecare pas al acestui subproces, un subproces de calcul pentru determinarea sumei din formula de calcul pentru elementul c_{ij}).

b) *Mulțimea variabilelor* (*V*)

Variabilele definite de raționamentul R, care vor fi utilizate în procesul de calcul P, sunt:

- k - puterea;
- n - ordinul matricei A;
- $A = \|a_{ij}\|_{i,j=1,n}$ - matricea inițială;
- $X = \|x_{ij}\|_{i,j=1,n}$ - matricea rezultat;
- $S = \|s_{ij}\|_{i,j=1,n}$, $P = \|p_{ij}\|_{i,j=1,n}$ - matrici de lucru.

c) *Proces[ul] de calcul* (*P*) *și reprezentarea algoritmului*

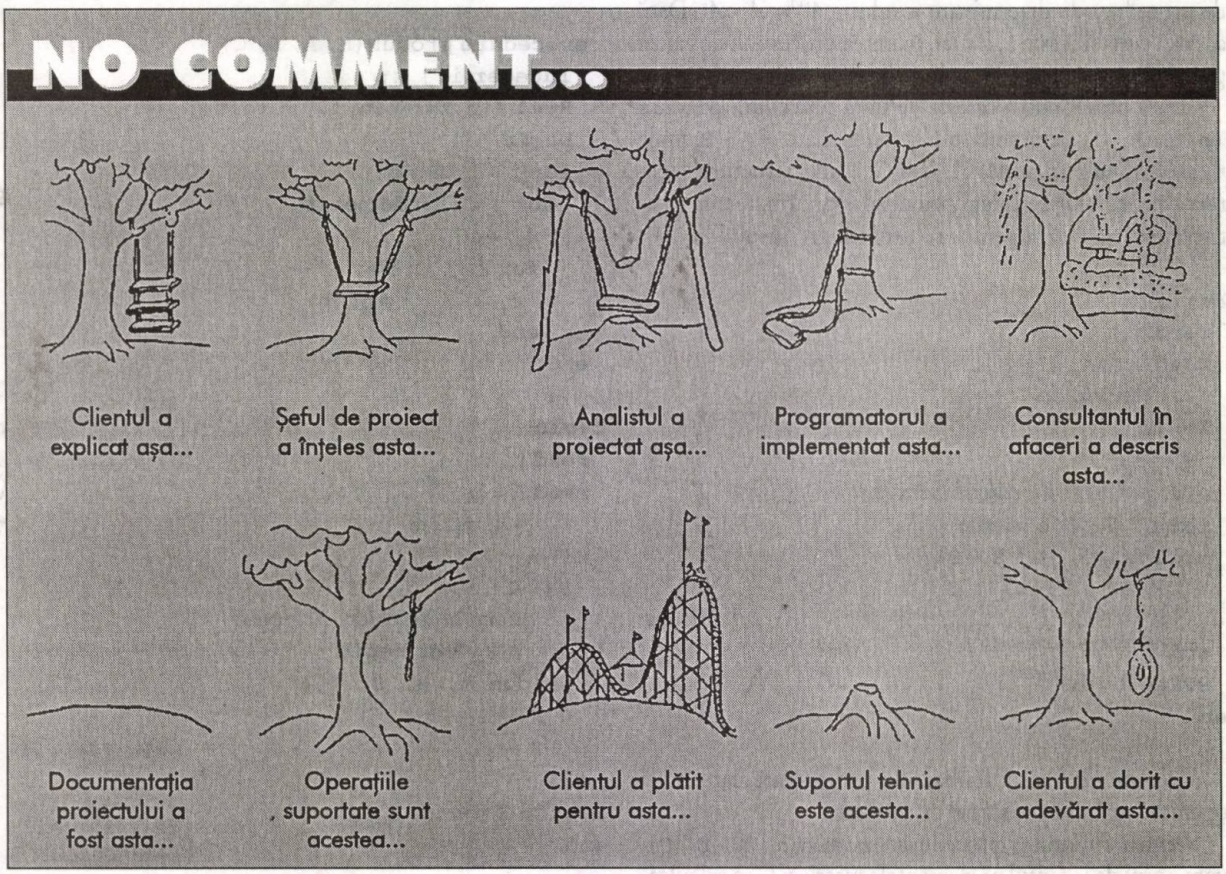
În cont[inuare] prezentăm algoritmul de rezolvare (procesul de calcul) a problemei:

```
Algorithm Suma_puterilor;
 Integer n, k;
 Real A(30,30), X(30,30),
      S(30,30), P(30,30);

 procedure produs(n, A, B, C)
  Integer i,j,k;
  Real A(n,n), B(n,n), C(n,n);
  begin
  for i = 1, n do
   for j = 1, n do begin
    c_ij ← 0;
    for k= 1, n do
     c_ij ← c_ij + a_ik · b_kj;
    end
 end

begin
 read n,k;
 read A = ||a_ij||_i,j=1,n;
  // inițializări
 X ← A;
  S ← A;
  // procesul de calcul principal
 for i = 2, k do begin
  produs(n, A, S, P);
    X ← X + P;
    S ← P;                          // salvare produs
 end
 write X = ||x_ij||_i,j=1,n;
end
```

Bibliografie

1. Albeanu Gr., *Algoritmi și limbaje de programare*, Universitatea "Spiru Haret", Editura România de Mâine, București, 2000
2. Apostol C., Roșca I. Gh., Roșca V., Ghilic-Micu B., *Introducere în programare. Teorie și aplicații*, Editura București, 1993
3. Georgescu H., *Programare concurentă. Teorie și aplicații*, Editura Tehnică, București, 1996
4. Kinston J. H., *Algorithms and Structures Design. Correctness, Analysis*, Addison Wesley, Longman, 1998
5. Mitrana V., *Provocarea algoritmilor. Probleme pentru concursurile de informatică*, Editura Agni, București, 1994
6. Odăgescu I., *Optimizarea algoritmilor*, Editura Militară, București, 1991
7. Perjeriu E., Văduva I., *Îndrumar pentru lucrări de laborator la cursul de Bazele Informaticii*, Universitatea din București, 1986
8. Popovici C., Georgescu H., State L., *Bazele Informaticii*, vol. I, Universitatea din București, 1990
9. Skiena S., *The Algorithm Design Manual*, Springer Verlag, New York, Inc., 1998
10. Vlada M., *Informatica*, Universitatea din București, Editura Ars Docendi, București, 1999
11. Vlada M., *Poligoane stelate. Problema lui Hopf și Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348
12. Cristea V., Giumale C., Kalisz E., Păunoiu Al., *Limbajul C standard*, Editura Teora, București, 1992
13. Zaharia M. D., *Structuri de date și algoritmi. Exemple în limbajele C și C++*, Editura Albastră, Cluj-Napoca, 2002
14. Popovici M. D., Popovici M. I., *C++. Tehnologia orientată spre obiecte. Aplicații*, Editura Teora, București, 2000
15. www.math.gatech.edu/~thomas/FC/fourcolor.html
16. mathworld.wolfram.com/Four-ColorProblem.html
17. spicerack.sr.unh.edu/~student/tutorial/fourColor/FourColor.html
18. mathcentral.uregina.ca/RR/database/RR.09.97/fisher1.html
19. www.uccs.edu/~asoifer/book5.html
20. www.uccs.edu/~asoifer/solve98.html
21. www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg
22. mathworld.wolfram.com/Algorithm.html
23. www.cs.rit.edu/~atk/Java/Sorting/sorting.html
24. faculty.juniata.edu/rhodes/cs2/ch129.htm
25. www.scism.sbk.ac.uk/law/Section5/chap1/ s5c1p2.htm
26. www.ics.uci.edu/~eppstein/161/people.html
27. www.cs.pitt.edu/~kirk/algorithmcourses/
28. www.gnacademy.org/text/cc/
29. java.sun.com/docs/books/tutorial/java/concepts/
30. www.accu.org/acornsig/public/articles/oop_c.html
31. loki.cs.brown.edu:8081/webae/full.html

Domnul conf. dr. Marin Vlada este cadru didactic la Universitatea București și poate fi contactat prin e-mail la adresa vlada@chem.unibuc.ro.



Cine are carte, are parte!

... de înțelegerea rețelelor!

Cea mai bună introducere din lume în rețele de calculatoare
- complet actualizată pentru tehnologiile cheie de mâine -

Comand-o acum și dacă ești student primești și o reducere semnificativă!

Byblos srl, tel: 021-330.92.81, fax: 021-330.92.85, e-mail: dpastrama@byblos.ro
www.byblos.ro
Librăria Eminescu, Universitate, la subsol, tel: 021-312.72.65; 0721-927.499

www.agora.ro
www.pcmagazine.ro
www.eweek.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

FORTE COMPUTERS
Flexibility. Made in Romania

IBM®

sat
IT SOLUTIONS & SERVICES

hp®

net consulting

FUJITSU COMPUTERS SIEMENS

SPONSOR DE AUR

CCT
CABINETUL CĂLIN TATOMIR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516

Gazeta de informatica
Revista viitorilor specialiști în tehnologia informației
Decembrie 2003
Vol. 13/8
Lei 30.000

din sumarul numărului:

actual

pledoarie pentru

JAVA

...o cafea bună

Plus

concurs

Bursele Agora Ediția 2003/2004

actual

Justiție: Intel se luptă cu Intelprof

internet

PHP 4 - utilizarea claselor Java

Premieră: Parteneriat strategic Microsoft - Guvernul României
PC MAGAZINE
ROMÂNIA
www.pcmagazine.ro
SOLUȚII
linux
Discobolul
Primii pași în Linux
office
15 sfaturi pentru Excel
grafică
Cum să animezi o sigiă
TENDINȚE
Remodelarea PC-ului
Reportaj de la
Intel Developer Forum
Imagine accesibilă
TESTE COMPARATIVE
Plăci video
PC LABS
GHIDUL TĂU ÎN LUMEA TEHNOLOGIILOR
Concurs de soluții!
Microsoft Office
super premii
un PokelPC,
premiul I
un SmartPhone,
premiul II
sau un pachet Office Professional 2003.
premiul III
20
de tehnologii de urmărit
interfață grafică optimizată
conținut nou substanțial
încărcat 100%
Ghidul tău în lumea tehnologiei
o publicație Agora

Vol. 13, Nr. 8 /2003

gazeta de informatică

Redacția:

Director științific:
prof. dr. *Horia Georgescu*
Director de publicație:
lect. univ. dr. *Clara Ionescu*
Director Bursele Agora:
Mihai Scorțaru
Redactor-șef:
Claudiu Soroiu
Redactor-șef adjunct:
Carmen Șușman
Redactori:
prof. *Eugen Ionescu*
Mihai Stroe

Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
400091, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 0264/592.422, 0265/266.516
Fax: 0265/266.290
http://www.ginfo.ro
ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 0265/262.359

Editor: Agora Media SRL
Agora Media SRL, editor al publicațiilor "*eWeek România*", "*PC Magazine România*" și "*Gazeta de Informatică*", este membru al Biroului Român pentru Auditarea Tirajelor.
str. T. Vladimirescu nr. 63/9
540014 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Vânzare reclamă:
Mihai Bucuroiu
Denis Marioara
Marius Comărniceanu
Difuzare:
Florin Dinu
E-mail: difuzare@buc.agora.ro
Tel: 021/330.92.82, int. 134

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 021/330.92.81
Fax: 021/330.92.85

Sumar

actual.. PAG. 5

Pledoarie pentru JAVA
Mihai SCORȚARU

Un nou proces INTEL
Mihai SCORȚARU

probleme.. PAG. 9

Concursul național de programare "STELELE INFORMATICII"
~ enunțurile problemelor ~

soluții.. PAG. 17

ACM
Mihai STROE
~ soluțiile problemelor ~
~ propuse spre rezolvare ~
~ la etapa regională ~

concurs.. PAG. 22

Bursele Agora
Ediția 2003 / 2004
~ enunțurile problemelor ~
~ propuse spre rezolvare ~
~ la primele cinci etape ~
~ ale concursului ~

mate.. PAG. 26

Principiul CUTIEI LUI DIRICHELET
Mihaela OLTEAN

ALGORITMI - CALCULABILITATE
Horia GEORGESCU

tehno.. PAG. 36

VIRTUAL PC 2004 VS. VMWARE 4
Claudiu SOROIU
~ prezentare comparativă ~

internet.. PAG. 40

Pagini WEB cu PHP 4
Mihai SCORȚARU
Claudiu SOROIU
~ utilizarea claselor Java ~
~ prelucrarea imaginilor ~

babel.. PAG. 46

Algoritmi de ORDONANȚARE
Ilie VIERU
~ ordinea efectuării ~
~ operațiilor ~

COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS AGORA

Timpul și... LENEA

Timpul este foarte important pentru fiecare dintre noi. De noi depinde cât de eficient ne împărțim timpul și cum ne organizăm activitățile de lucru și de divertisment, pentru a avea un randament ridicat și pentru a nu fi extenuați. Trebuie să învățăm să ne împărțim timpul și să profităm de el la maximum.

În primul rând trebuie să stabilim ce anume avem de făcut și care sunt activitățile cu prioritate. Acestea fiind stabilite, trebuie să ne gândim că avem nevoie și de odihnă (relaxare) pentru ca să putem lucra la fel de bine ca la început. Adică, nu trebuie să lăsăm oboseala să ne doboare, nici stresul să ne chinuie. Cel mai bun ar fi un program echilibrat, în care *să îmbinăm munca și distracția* și să nu exagerăm cu nici una dintre acestea!

În legătură cu cele exprimate mai sus, voi deschide un subiect, cred eu, important pentru elevi, studenți și părinți. Majoritatea părinților, afirmația se bazează exclusiv pe cele observate de mine, își încurajează copiii prin *pune mâna și învață!* sau *toată ziua te-ai uitat la televizor și tu ar trebui să o petreci învățând!* mai ales atunci când copilul respectiv face o pauză. Înțelegem că bucuria părintelui constă în a-și vedea odrasla învățând, dar mai și obosește omul și vrea o pauză... Dacă se abuzează de această pauză, atunci nu am de ce să nu îi îndreptățesc pe părinți!

Timpul are multiple legături și cu alte lucruri care țin exclusiv de felul fiecăruia de a fi. Cele mai importante lucruri care "intră în contact direct" cu timpul nostru sunt: *voința*, *puterea de muncă*, *puterea de a depăși lenea*, *puterea de concentrare*, *organizarea muncii* (despre care am pomenit mai sus), *adaptabilitatea*, *toleranța*, *seriozitatea* și *conștiinciozitatea*.

De ce le consider a fi cele mai importante? Voi răspunde la această întrebare pentru fiecare în parte. *Voința*. Toți știm că dacă nu ai voință nu realizezi nimic. *Puterea de muncă*. Dacă un om are putere moderată de muncă, atunci nu trebuie să își asume responsabilități mai multe decât poate "duce". *Puterea de a depăși lenea*. Aceasta constă în voința noastră, bineînțeles, dar mai ales în înțelegerea că a face eforturi de începere a lucrului înseamnă a avea de câștigat! Ce câștigăm? Câștigăm experiență și, cu timpul, îndemânarea de a învinge lenea! *Puterea de concentrare*. Dacă reușim să ne concentrăm în momentul în care începem să facem ceva, înseamnă că vom putea lucra mai ușor și mai repede. Deci, suntem mai eficienți. Să nu uităm că pentru a putea să ne concentrăm trebuie să fim odihniți! *Organizarea muncii* și *adaptabilitatea* merg mână în mână. Munca ne-o organizăm în funcție de volumul ei, iar noi trebuie să ne adaptăm repede și ușor, mai ales atunci când, având mult de lucru, trebuie să trecem de la un domeniu la altul. *Toleranță*. Ei bine, această caracteristică nu se referă la a tolera pe altcineva... Se referă la cât de toleranți suntem cu noi înșine! Dacă suntem prea toleranți, atunci vom lucra încet și mult timp, dar dacă suntem mai aspri, atunci nu ne ridicăm de la masa de lucru pentru pauză cu una cu două. Aceste lucruri se întâmplă și dacă nu suntem destul de serioși și conștiincioși. Trebuie să fim capabili, suntem maturi sau cel puțin așa pretindem, deci este nevoie să acționăm ca atare. A fi *serios* și *conștiincios* înseamnă a munci cu sârguință și devotament, chiar cu plăcere, înseamnă a fi privit cu alți ochi de cei din jur, înseamnă a câștiga respectul celorlalți, înseamnă a fi lăudat și apreciat, înseamnă a fi mulțumit, înseamnă a fi hotărât să continui munca, înseamnă a dori performanța!

Da, aveți dreptate, am folosit de multe ori verbul *trebuie* până acum, dar așa am simțit că *trebuie* să mă exprim.

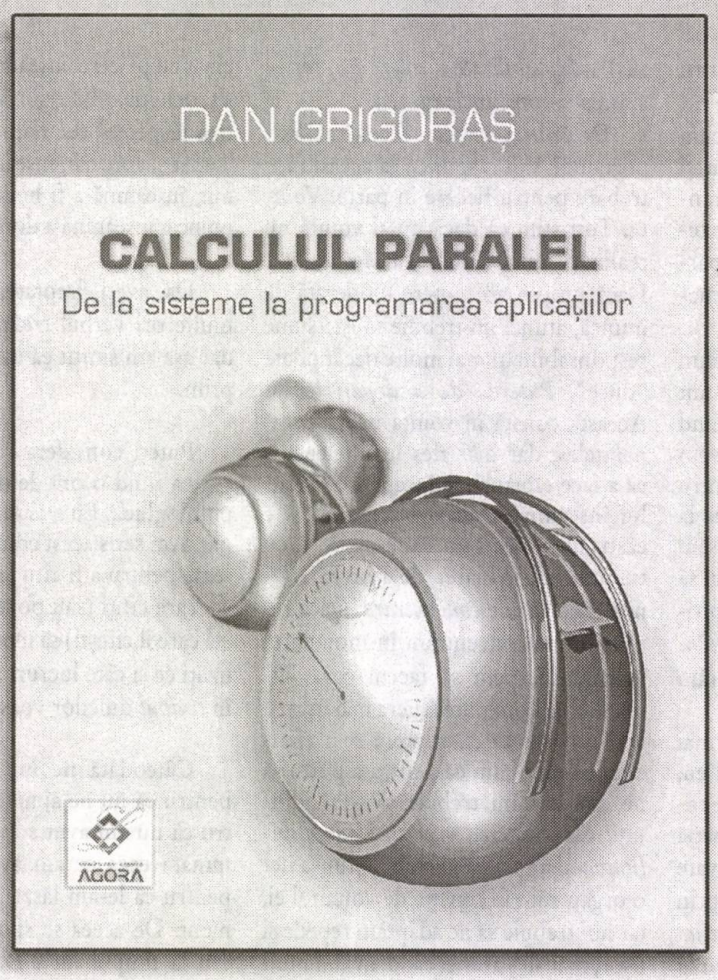
Puteți considera ce am scris mai sus ca fiind o oră de dirigenție sau o pologlohie... Eu nu am de ce să mă supăr. Am scris acest editorial pe această temă pentru a fi citit și pentru ca voi, cei care citiți (sau poate pentru ca tu, cel care îl citești) să meditați și să analizați ce și câte lucruri ați fi putut face în *timpul* dulcilor voastre leneviri.

Câteodată ne luptăm cu timpul pentru că nu ne ajunge... dar nu pentru că nu am avut suficient ca să terminăm ceea ce am avut de făcut, ci pentru că le-am lăsat pe ultimul moment. De aceea se spune "fiecare lucru la timpul lui". Da, știu la ce va gândiți acum "ce poți face azi, nu lăsa pe mâine, lasă pe poimâine că poate nu mai trebuie". Ha! Ha! Dar vă răspund "lenea e cucoană mare"!

Oricum timpul reprezintă o resursă suficientă pentru fiecare dintre noi. De noi depinde cât de bine ne folosim de ea, cât avem de câștigat mișcându-ne repede. Timpul nu reprezintă o constrângere, dacă nu îl transformăm în dușman. Asta ne-ar putea distruge!

Multe lucruri pot fi recuperate, dar, din păcate, nu și timpul!

Carmen Șușman



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

Pledoarie pentru JAVA

Mihai Scorțaru

Nu știu câți dintre voi își aduc aminte de un articol cu titlul "Pledoarie pentru C", scris de un membru al Lotului Național de Informatică, în urmă cu aproape zece ani. În articolul respectiv, se spunea că la concursurile de programare, majoritatea concurenților folosesc limbajul Pascal, ignorând facilitățile oferite de C. Eu nu mă voi referi la concursuri, ci la cursuri... Mai exact la limbajele de programare predate în școli. Aici avem Pascal sau C, dar poate ar fi mai bine să fie Java...

În *România* la liceu se predă *Pascal* și C++, folosindu-se, de obicei, mediile *Borland*. În facultate, studentul învață și alte limbaje de care are nevoie pentru a deveni programator sau cercetător.

După ce termină facultatea, studentul român cunoaște o mulțime de limbaje, care mai de care mai ciudate și mai nefolositoare, dar știe și câteva pe care le poate folosi dacă va deveni programator.

Din păcate, această afirmație este adevărată doar din punct de vedere teoretic. Practic, studentul român nu învață nimic la facultate. Aproape toți programatorii susțin că au învățat singuri...

Să ne gândim puțin la un elev de clasa a IX-a al cărui vis este să devină programator. În liceu va învăța din greu să implementeze algoritmi care mai de care mai complicați, folosind limbajele mai sus amintite și va învăța să lucreze cu baze de date.

După ce termină liceul își dă seama că nu știe nimic din ceea ce are nevoie. Știe rezolva tot felul de probleme dar nu știe să creeze o aplicație simplă, cu o interfață prietenoasă. Așa că... merge la facultate ca să învețe acolo.

Așadar, un elev de la clasele de matematică-informatică (intensiv sau nu) este obligat să urmeze și o facultate. Liceul nu îl pregătește suficient pentru meseria pe care și-o dorește deși, în opinia mea, așa ar fi normal.

Nu mi se pare a fi hazardată afirmația potrivit căreia un absolvent de liceu poate deveni programator (cu acte în regulă) fără să fie și "om cu facultate". De ce? Pur și simplu pentru că poate! Știm că o mulțime de studenți lucrează pe post de programatori deși nu sunt "programatori cu acte în regulă". Cu siguranță, au învățat singuri... dar au învățat, deci se poate!

Și de ce să nu fie ajutați toți? Așa ar trebui, veți spune... De ce un elev obișnuit, care termină liceul, să nu poată deveni programator? Nu știu! Dar, deocamdată nu poate!

O problemă de mentalitate

Era demult,
Era frumos,
Era ca-ntr-o poveste,
Era atunci,
Era și-a fost,
Era și nu mai este!

Era demult o perioadă când la liceu în clasa a IX-a se învăța *QBasic*. Aceasta se întâmpla în prima jumătate a anilor '90. Apoi a venit cineva și a zis că nu se mai poate! Și așa, elevii de clasa a IX-a au început direct cu *Pascalul Wow!!!*. Nu mai e așa... *Basic*-ul e prea vechi. Pascalul e mult mai bun, *C*-ul e și mai bun, *C++*-ul e mai ceva, iar *Java* e... undeva pe aici...

S-a schimbat... Deci.. se poate schimba! A fost greu? Da! De ce? Fiindcă le va fi mult mai greu elevilor să învețe *Pascal* decât *Basic*!

Acesta era răspunsul oficial! Din nefericire, cel real era: Fiindcă le va fi mult mai greu profesorilor să predea *Pascal* decât *Basic*!

Și totuși... s-a schimbat... *Basic*-ul a fost uitat și au apărut *Pascal* și *C/C++* (pentru clasele mai mari).

Apoi s-a pus problema ca în clasa a IX-a să se înceapă direct cu *C*-ul. Uitatul "război" *Basic* vs. *Pascal* s-a transformat în *Pascal* vs. *C*. De ce? Fiindcă le va fi mult mai greu elevilor să învețe *C* decât *Pascal*!

Țin minte remarca unui profesor de informatică atunci când am susținut că limbajele de programare predate în licee trebuie schimbate mult mai des: "Profesorul de mate' iese din facultate și știe tot ce are nevoie pentru a preda matematică la liceu cincizeci de ani de acum încolo. Are salar cât al meu. El nu mai trebuie să învețe nimic, eu de ce să trebuiască să învăț?". Poate fiindcă sunteți profesor de informatică și nu de matematică, m-am gândit eu... Dar

cuvintele nu au fost rostite din... motive de politețe. De ce apar aici? Așa... fiindcă trebuie să apară!

Și totuși, și atunci s-a făcut o jumătate de pas. Programa nu mai precizează limbajul de programare. Profesorul poate alege... Poate preda *Pascal* sau *C*. Unii susțin chiar faptul că poate preda orice: *Algol*, *Fortran*, *Java*, *C#* sau orice altceva îi trece prin minte. Nu prea îmi vine să cred... Din câte știu eu, la *Bacalaureat*, există o variantă *Pascal* și o variantă *C*. Doar atât!

Acești pași (și jumătăți de pași) au fost utili... Dar nu ajunge! Programa actuală duce la crearea unor vârfuri (care obțin premii la olimpiade) și a unei mulțimi de neștiutori care nu sunt în stare să deosebească un algoritm de determinare a permutărilor de un algoritm de determinare a elementului minim dintr-un șir. Și ce dacă nu știu? Cineva îi va învăța să deseneze interfețe în *Delphi* și vor deveni programatori.

Mi se pare inutil acest efort de a îi învăța pe elevi algoritmică... E necesară, într-adevăr, dar nu atât de necesară precum cunoașterea unui limbaj de programare modern.

Problema este, cu siguranță, una de mentalitate... Se aplică principiul: "*Rezultate de calitatea care se nimerește, dar cu eforturi minime*". Cu o astfel de mentalitate, elevul va trece prin liceu fără să cunoască nimic util. Cu o astfel de mentalitate, profesorii se înscriu la doctorat propunându-și ca temă a lucrării metodele de sortare! Credeți-mă! Am fost de față. Aceasta a fost tema lucrării pe care a propus-o un profesor care predă informatică la liceu!

Și acum... să vedem de ce îi este mai greu unui elev să învețe *Java* decât să învețe *C* sau *Pascal*. *For*-ul e tot *for*, *while*-ul e tot *while*, *if*-ul tot *if*. Deci, structurile de control care trebuie explicate sunt aceleași. Subprogramele sunt tot subprograme, deci nu avem nici aici probleme. Sintaxa e mai greoaie în *Java*! Posibil, deși practic identică celei din *C*! Numai că în *Java* avem obiecte și trebuie să fie explicate noțiuni complicate cum ar fi moștenire, polimorfism, etc. Chiar trebuie? Bineînțeles! Dar nu la început!

La început nu trebuie să se predea nimic mai complicat decât ceea ce se predă acum. Pe parcurs, se pot introduce paradigmele programării orientate obiect. Ceea ce nu se are în vedere este faptul că elevul nu știe nimic. De ce să credem că îi va fi mai ușor să înțeleagă și să utilizeze principiile programării structurate decât pe cele ale programării orientate obiect? Numai fiindcă nouă ni se pare că programarea structurată e mai ușor de învățat? Cu siguranță cei care stăpânesc ambele tehnici nu au probleme în a le utiliza. Nu li se pare că una este mai ușoară decât alta. Ambele tehnici conțin elemente de bază care trebuie înțelese.

Permiteți-mi să afirm că principiile programării orientate obiect nu sunt mai greu de înțeles decât principiile programării structurate. Și sunt mult mai utile! De ce nu sunt mai greu de înțeles? Fiindcă elevul nu are alte cunoștințe! Nu e obișnuit să lucreze pe baza unor principii, la fel ca profesorii. Orice cunoștințe noi sunt la fel de ușor de acumulat; totul depinde de talentul și seriozitatea celui care transmite cunoștințele respective. Așadar, totul depinde de profesori. Cei care au salarii mizere și abia așteaptă să scape de învățământ. Dar dacă sunt plătiți, atunci să își facă treaba. Dacă nu să plece!

Veți spune că vor pleca toți... Poate așa și trebuie... Poate că dacă nimeni nu va mai preda, salariile vor crește... Poate... Oricum, faptul că ești prost plătit nu reprezintă o scuză pentru a nu-ți face meseria. Dacă meseria ta implică învățarea permanentă de lucruri noi, atunci trebuie să le înveți... Dacă scopul meseriei tale este să îi ajuți pe ceilalți, atunci ajută-i, oferă-le ceea ce au nevoie!

Mă opresc aici cu generalitățile... În continuare voi încerca să arăt de ce consider că limbajul *Java* este, într-adevăr, o variantă demnă de luat în seamă.

Mediul de programare

Pentru *Pascal* se folosește acum *Borland Pascal* 7.0. Pentru *C/C++* avem *Borland C++* 3.1. Poate sunt alte versiuni, poate în unele cazuri sunt chiar alte medii. Totuși, în majoritatea cazurilor, acestea sunt cele folosite.

Puțini își pun problema, dar aceste pachete *NU* sunt gratuite. Chiar dacă *Borland* oferă gratuit în scopuri educaționale anumite aplicații, în majoritatea școlilor românești nu există licențe.

Pentru *Java* avem o mulțime de medii de programare gratuite. Chiar avem de unde alege!

În plus, pentru mediile *Borland* avem nevoie de *Windows* (sau *DOS*). Și sistemul de operare costă. Chiar dacă acum, împreună cu ultimele dotări, în școli au ajuns și licențe *Windows*, acestea nu există peste tot!

Dacă utilizăm *Java*, putem să folosim *Linux*, ceea ce rezolvă problema pirateriei.

În plus, mediile de programare *Java* oferă o serie de facilități la care elevii de acum nici măcar nu visează. Pe lângă *code-completion* (facilitate care ar fi suficientă pentru ca să renunțăm la un mediu de programare datorită lipsei ei), avem tot felul de avantaje cum ar fi semnalarea erorilor de compilare pe măsură ce editățm codul, includerea automată a librăriilor pe care le utilizăm etc.

Avantaje

Cel mai important avantaj este faptul că *Java* este un limbaj care s-a impus și nu o simplă modă. Cu siguranță va supraviețui câțiva ani buni. Așadar, cu cât este adoptat mai repede, cu atât mai bine... Ar fi chiar caraghios să apară în programele școlare atunci când începe să nu mai fie folosit.

Un alt avantaj este faptul că există o sumedenie de tehnologii *Java*, create pe baza nucleului limbajului. Chiar dacă nu se va putea preda în liceu *RMI*, *JSP* etc., unui elev îi va fi mult mai ușor să înțeleagă aceste tehnologii dacă deține cunoștințele de bază.

Java este un limbaj obiectual, la fel ca marea majoritate a celorlalte limbaje folosite acum pentru crearea de produse *software*. Dacă este cunoscut un astfel de limbaj, celelalte sunt mult mai ușor de învățat atunci când e nevoie de ele!

Dezavantaje

Nu voi enumera nici un dezavantaj deoarece nu consider nici unul a fi demn de luat în seamă. Aștept replicile tuturor celor care nu sunt de acord cu ceea ce am afirmat anterior. Așadar, dragi profesori, aici *GInfo* vă oferă dreptul la replică.

Câteva exemple

În cadrul acestei secțiuni voi prezenta câteva scurte exemple prin care voi încerca să arăt "cât de bună e cafeaua".

Operații cu numere mari

Aceste operații sunt un chin pentru toți cei care participă regulat la concursurile de programare. Foarte des e nevoie să implementezi așa ceva. Iar dacă mai apare și o împărțire, atunci concursul devine un coșmar.

Datorită faptului că *Java* este un limbaj obiectual și unul dintre avantajele programării orientate obiect este reutilizabilitatea codului, în *Java* nu avem această problemă. Cineva a implementat deja aceste operații și noi nu trebuie decât să le utilizăm.

Iată cum aceste operații complicate sunt implementate în *Java* folosind doar câteva linii de cod:

```
import java.math.BigInteger;

public class NumereMari {
  public static void main(String[] args) {
    BigInteger a = new BigInteger("2458");
    BigInteger b = new BigInteger("13");
    System.out.println(a + " + " + b + " = "
                                  + a.add(b));
    System.out.println(a + " - " + b + " = "
                             + a.subtract(b));
    System.out.println(a + " * " + b + " = "
                            + a.multiply(b));
    System.out.println(a + " / " + b + " = "
                             + a.divide(b));
    System.out.println(a + " % " + b + " = "
                            + a.remainder(b));
  }
}
```

Cam atât e nevoie să scriem... Dar nu se oprește aici! Avem și operații cu numere reale... Iată!

```
import java.math.BigDecimal;

public class NumereRealeMari {
  public static void main(String[] args) {
    BigDecimal a = new BigDecimal("2458");
    BigDecimal b = new BigDecimal("13");
    System.out.println(a + " + " + b + " = "
                                  + a.add(b));
    System.out.println(a + " - " + b + " = "
                             + a.subtract(b));
    System.out.println(a + " * " + b + " = "
                            + a.multiply(b));
    System.out.println(a + " / " + b + " = "
    + a.divide(b, 5, BigDecimal.ROUND_DOWN));
  }
}
```

Arhivare

Puteți spune că, totuși, după câteva ore, ați fi reușit oricum să implementați operațiile cu numere mari, chiar și pentru numere reale... Probabil că aveți dreptate dar, oricum, ar fi câteva ore pierdute.

Totuși, *Java* vă ajută să economisiți zile sau chiar luni. Cât credeți că v-ar lua să transformați un fișier într-o arhivă *ZIP*? Probabil că veți avea nevoie de câteva zile doar pentru a studia formatul fișierului. În *Java*, veți reuși să arhivați fișiere în două-trei minute. Următoarea secvență este, cred eu, sugestivă:

```
import java.util.zip.ZipOutputStream;
import java.util.zip.ZipEntry;
import java.io.FileOutputStream;
import java.io.FileInputStream;

public class ArhivareZIP {
  public static void main(String args[])
                                  throws Exception {
    ZipOutputStream zo = new ZipOutputStream
      (new FileOutputStream("fisier.zip"));
    ZipEntry e = new ZipEntry("fisier.txt");
    zo.putNextEntry(e);
    FileInputStream in = new
          FileInputStream("fisier.txt");
    byte b[] = new byte[16384];
    int i;
    while ((i = in.read(b)) > 0)
        zo.write(b, 0, i);
    zo.flush();
    zo.closeEntry();
    zo.close();
  }
}
```

Încheiere

Ar trebui să trag câteva concluzii acum, dar nu o voi face... Prefer să aștept reacțiile dumneavoastră. Așadar, vă rog să vă exprimați opinia în legătură cu această "porție de cafea".

Evident, nu am vrut să încep aici o prezentare a limbajul *Java*. Secvențele de cod au fost incluse doar pentru a ilustra cât de ușoară este realizarea unor operații care necesită multe ore sau zile de muncă dacă se folosesc limbajele predate acum în licee.

Este important să vă faceți cunoscut punctul de vedere. Nu trebuie să fiți de acord cu ce este scris aici... dar nu puteți fi de acord nici cu statul pe loc în învățământul românesc de informatică. *Așteptăm scrisorile voastre!*

Un nou proces INTEL

Mihai Scorțaru

Am citit recent în "Curierul Național" o știre care mi-a atras atenția. Se pare că celebrul producător Intel a intentat un proces unei companii din România... Motivul: denumirea companiei românești conține și grupul de caractere "intel". În continuare prezentăm câteva detalii referitoare la acest "eveniment" care, pentru noi, pare destul de interesant.

Compania din România care a fost acționată în instanță de *Intel* se numește *Intelprof*. Se pare că acționarii acestei firme nu au făcut cea mai bună alegere atunci când au ales numele deoarece, se pare, această denumire nu convine companiei americane.

Motivul invocat de *Intel* este destul de interesant: existența unei companii cu numele *Intelprof* face să se "diminueze valoarea de piață a mărcii *Intel*".

Iată cum un moment de lipsă de inspirație îți poate aduce neplăceri. Chiar dacă semnificația grupului de litere "*intel*" este diferită în numele celor două companii, forma în care apare această succesiune de litere în denumiri este identică.

Pentru compania americană *Intel* este prescurtarea de la *integrated electronics*, în timp ce pentru compania de la noi, *intel* este prescurtarea de la *inteligență*.

De fapt, *Intelprof* vrea să însemne *inteligență* și *profesionalism*. Chiar dacă inteligența este necesară pentru crearea de circuite electronice integrate, o altă legătură nu prea poate fi stabilită între cele două companii.

Cu toate acestea, se pare că *Intel* ține atât de mult la denumirea sa, încât ajunge să considere că simpla prezență a aceleiași secvențe de caractere într-o altă denumire poate fi privită ca o amenințare la adresa companiei americane.

Intelprof

Compania românească *Intelprof* a fost înființată în 1993 de către *Gheorghe* și *Angela Dobre*. Așadar, se pare că *Intel* a avut nevoie de zece ani pentru a realiza că doi români au creat o firmă cu scopul de a submina valoarea mărcii.

Inițial, compania românească oferea cursuri de folosire a calculatoarelor pentru ca apoi obiectul principal de activitate să devină predarea de cursuri de certificare *Microsoft* pentru specialiști *IT*.

Denumirea *Intelprof* a fost înregistrată legal la *Oficiul de Stat pentru Invenții și Mărci* și nimeni nu a avut nimic de obiectat până acum.

Litigiul

Procesul dintre *Intel Corporation* și *Intelprof* are loc la *Tribunalul Municipiului București*. Este un proces fără precedent în *România*, motiv pentru care decizia finală va avea o importanță deosebită. Conform practicilor judiciare, această decizie va fi luată în considerare la toate procesele similare care se vor desfășura în viitor.

Intel cere instanței române constatarea faptului că *folosirea mărcii Intelprof încalcă drepturile exclusive ale Intel*. Mai mult, compania românească este acuzată de concurență neloială. Potrivit legislației din *România*, concurența neloială constă în *întrebuințarea unei firme, unei embleme, unor desemnări speciale sau a unor ambalaje de natură a produce confuzie cu cele folosite legitim de alt comerciant*.

Ce putem înțelege de aici? Mai nimic, cred eu... Poate că este o manevră care să atragă atenția asupra companiei *Intel* într-o țară în care marea majoritate a utilizatorilor casnici preferă procesoarele *AMD* celor de la *Intel*. Poate... sau poate nu... Poate numele contează atât de mult încât nu se poate accepta existența unor companii a căror denumire să sugereze o posibilă legătură cu *Intel*.

Oricum, cert este faptul că *Intel* dorește: *anularea mărcii Intelprof, prin radierea sa din Registrul Național al Mărcilor, interzicerea utilizării pe viitor a numelui Intelprof, constatarea "actelor de concurență neloială" săvârșite de pârâtă împotriva reclamantei și publicarea, în presa scrisă, pe cheltuiala Intelprof, a dispozitivului hotărârii care va fi pronunțată, în termen de cel mult 10 zile de la rămânerea definitivă și irevocabilă a acesteia*.

Poate urmează și alții...

Potrivit articolului publicat în "*Curierul Național*", în *România* există peste 300 de companii în a căror denumire apare șirul de caractere *intel*. Este greu de crezut că *Intel* se va lupta în justiție cu toate aceste companii. Mai curând, s-ar putea ca *Intel* să își imagineze că aceste companii se vor "speria" și își vor schimba denumirea. *Intel* uită totuși că: "suntem în *România*"...

22-30 noiembrie 2003, București

Concursul Național de Programare "STELELE INFORMATICII"

Mihai Stroe

În perioada 22-30 noiembrie 2003 a avut loc la București prima ediție a Concursului Național de Programare "Stelele informaticii", organizat de Liceul Internațional de Informatică din București. În continuare vă prezentăm detalii referitoare la acest concurs și enunțurile celor 12 probleme propuse spre rezolvare.

La acest concurs, redacția *GInfo* a avut un reprezentant care a fost direct implicat în organizarea concursului și care a fost în măsură să descrie în detaliu întreaga acțiune.

Competiția a reunit la start unii dintre cei mai valoroși elevi care au obținut premii la diferite olimpiade.

A existat o perioadă de înscriere a participanților; au avut prioritate elevii care au obținut rezultate foarte bune la *ONI* 2003 și membrii lotului național lărgit din acest an.

Acest concurs a dat startul pregătirii pentru olimpiadele de informatică din 2004, fiind urmat de concursurile și activitatea de pregătire organizată demarate în luna decembrie: *Bursele AGORA* și *.Campion* (*punct Campion*). Împreună cu celelalte concursuri regionale și naționale deja existente se obține astfel un calendar competițional complet, care asigură condițiile obținerii unor rezultate valoroase la concursurile internaționale.

Participanții la acest concurs au fost împărțiți în două grupe de vârstă: elevi ai claselor a IX-a și a X-a, respectiv ai claselor a XI-a și a XII-a. Concursul de la clasele a IX-a și a X-a a avut loc în zilele de 22 și 23 noiembrie, iar cel de la clasele mari în 29 și 30 noiembrie. Pentru fiecare grupă au existat 2 zile de concurs. În fiecare zi au fost propuse spre rezolvare câte trei probleme, similare problemelor folosite la olimpiadele naționale de informatică; timpul de lucru a fost de 4 ore la clasele a IX-a și a X-a, respectiv 5 ore la clasele a XI-a și a XII-a.

Pentru fiecare grupă s-a folosit un set unic de subiecte, clasamentul făcându-se separat pe clase.

Primii clasați au primit premii constând în aplicații *software* oferite de *SOFTWIN* și premii în bani din partea *Liceului Internațional de Informatică* din București. S-au acordat un premiu I, un premiu II, un premiu III și între una și patru mențiuni la fiecare clasă, în funcție de punctajele obținute.

De asemenea, liceul a asigurat, din fonduri proprii, cazarea pentru o parte din concurenții din provincie, masa pentru toți concurenții și a recompensat financiar membrii juriului.

Din partea liceului, principalul organizator a fost domnul profesor *Ali Yuksel*, care a supervizat și a asigurat buna desfășurare a concursului, punând la dispoziție toate resursele necesare. Personal apreciez că organizarea a fost excelentă, lucru confirmat și de către concurenți.

Concursul s-a bucurat de sprijinul *Inspectoratului Școlar al Municipiului București.*

Juriul, condus de domnul *prof. univ. dr. Adrian Atanasiu*, a avut în componență studenți, foști olimpici: *Mihai Stroe*, *Marius Andrei*, *Mugurel-Ionuț Andreica*, *Radu Berinde* și *Bogdan Stroe*. De asemenea, din juriu a făcut parte domnul profesor *Osman Ay* de la *Liceul Internațional de Informatică din Constanța*, propunător de probleme la ultimele ediții ale etapei internaționale a concursului studențesc *ACM*.

Concurenții au avut la dispoziție sisteme de calcul performante, cu aceeași configurație și au putut rezolva proble-

mele folosind mediile de programare *Borland Pascal*, *Borland C*, *gcc* și *freepascal*. Corectarea s-a realizat pe sistemul de operare *Linux* cu ajutorul compilatoarelor *gcc* și *freepascal*; s-a folosit sistemul de evaluare de la *BOI* 2003. Pregătirea sistemelor și asistența tehnică au fost asigurate de personalul liceului, ajutat de elevi olimpici (*Alexandru Moșoi*, *Alexandru Radu*).

Concursul a fost considerat un succes. *Liceul Internațional de Informatică* își propune organizarea anuală a acestei competiții, aproximativ în aceeași perioadă. Detaliile organizării ediției următoare vor fi anunțate în timp util, înainte de organizarea *ONI* 2004.

În prima grupă de vârstă (clasele a IX-a și a X-a) au participat 22 de concurenți dintre care unul este elev în clasa a VIII-a (*Bogdan Stoica* - a obținut mențiune), 6 sunt elevi în clasa IX-a, iar 15 sunt elevi în clasa a X-a.

Clasamentul la clasa a IX-a este următorul:

Premiul 1: Valentin Stanciu (București)	*340*
Premiul 2: Cristina Stancu-Mara (București)	*200*
Premiul 3: Bogdan Ionescu (București)	*110*
Mențiune: Bogdan Stoica (București, cls. VIII)	*90*
Mențiune: Andrei Gönczi (Arad)	*80*

Clasamentul la clasa a X-a este următorul:

Premiul 1: Mircea Pașoi (Ploiești)	*460*
Premiul 2: Daniel Păsăilă (Suceava)	*430*
Premiul 3: Mihnea Giurgea (Ploiești)	*410*
Mențiune: Adrian Vladu (București)	*330*
Mențiune: Bogdan Tăutu (București)	*270*
Mențiune: Alex Florescu (București)	*260*

Pentru cea de-a doua grupă de vârstă (clasele a XI-a și a XII-a) au participat 28 de concurenți dintre care 14 sunt elevi în clasa XI-a, iar 14 sunt elevi în clasa a XII-a.

Clasamentul la clasa a XI-a este următorul:

Premiul 1: Leonard Crestez (Brăila)	*300*
Premiul 2: Iolanda Popa (Iași)	*280*
Premiul 3: Dan Fechete (Suceava)	*240*
Mențiune: Sorin Stancu-Mara (București)	*220*

Clasamentul la clasa a XII-a este următorul:

Premiul 1: Silviu Gănceanu (București)	*360*
Premiul 2: Alexandru Moșoi (București)	*310*
Premiul 3: Dan Ghinea (București)	*270*
Mențiune: Cristian Cutocheraș (București)	*250*
Mențiune: Mircea Digulescu (București)	*230*
Mențiune: Emilian Miron (Bacău)	*230*
Mențiune: Marius Nicolae (Slatina)	*220*

Menționăm că primii clasați la clasele a IX-a și a X-a au primit invitația de a participa și la concursul claselor a XI-a și a XII-a, fără a fi luați în calcul la acordarea premiilor. O parte dintre ei au obținut rezultate notabile. Astfel, *Mircea Pașoi* a obținut 270 puncte, *Daniel Păsăilă* - 250 puncte, iar *Bogdan Tăutu* - 220 puncte (pentru a face comparație cu premianții claselor a XI-a și a XII-a).

Prezentăm în continuare problemele propuse spre rezolvare la acest concurs, urmând ca în numărul următor să vă oferim indicații de rezolvare pentru aceste probleme.

Clasele a IX-a și a X-a

P080301: Cuvinte

Mihai Stroe

Doi prieteni, *Marius* și *Andrei*, s-au gândit la un joc.

Marius scrie pe o foaie un șir de *N* numere. Sub fiecare număr, el scrie câte o literă: sub primul număr litera *A*, sub al doilea număr litera *B*, și tot așa, în ordine lexicografică. *Marius* și *Andrei* folosesc un alfabet cu câteva mii de litere, cunoscut numai de ei și care începe cu literele de la *A* la *Z* (pentru a putea fi folosit și în comunicarea cu alți oameni). Literele sunt deci folosite ca indici pentru numerele din șir.

Andrei caută apoi toate "cuvintele" posibile care respectă următoarele condiții:

- un cuvânt reprezintă un șir de litere, ordonat lexicografic;
- numerele din șir, corespunzătoare literelor dintr-un cuvânt și scrise în ordinea dată de acestea, sunt în ordine strict crescătoare.

De exemplu, pentru șirul 2 1 3 5 4, scriind dedesubt literele *A B C D E*, câteva dintre cuvintele valide sunt *AC*, *ACD*, *ACE*, *AD*, *BE* etc., dar *AB*, *ED* sau *BDE* nu sunt cuvinte valide.

Apoi, *Andrei* alege dintre aceste cuvinte pe cele de lungime maximă și le scrie în ordine lexicografică. Pentru exemplul de mai sus, acestea sunt *ACD*, *ACE*, *BCD* și *BCE*. Dintre aceste cuvinte de lungime maximă, el i-l spune lui *Marius* pe al *K*-lea în ordine lexicografică. Dacă *Andrei* spune corect (și repede) cuvântul acesta, el câștigă jocul și îl pierde în caz contrar.

Andrei câștiga întotdeauna și era foarte mândru de el, până într-o bună zi, când *Marius* i-a scris un șir de 67 de numere... I-ar fi fost cam greu lui *Andrei* să facă tot ce făcea de obicei, așa că s-a gândit să scrie un program care să-i dea direct rezultatul. Pentru asta, a apelat la un profesionist.

Scrieți un program care determină cuvântul cerut și îl ajută pe *Andrei* să câștige jocul.

Date de intrare

Fișierul de intrare **cuvinte.in** conține pe prima linie două numere *N* și *K*, separate printr-un spațiu, reprezentând numărul de numere din șir, respectiv numărul de ordine al cuvântului cerut. Pe a doua linie se află *N* numere întregi separate prin câte un spațiu, numerele scrise de *Marius* pe foaie.

Date de ieșire

În fișierul **cuvinte.out** se va scrie cuvântul cerut. Deoarece nu se cunosc literele care urmează după *Z* în alfabetul celor doi, în locul literelor cuvântului se vor scrie numerele

de ordine ale acestora în alfabet, separate prin câte un spațiu. Astfel, de exemplu, cuvântul *ACZ* ar fi scris ca 1 3 26.

Deci, în fișier se vor scrie, separate printr-un spațiu, numere întregi care reprezintă numerele de ordine în alfabet ale literelor celui mai lung cuvânt cu numărul de ordine *K*, format după regulile din enunț.

Restricții și precizări

- N este număr natural și $1 < N \leq 200$;
- K este număr natural și $1 \leq K \leq 2.000.000.000$;
- numerele din șir sunt întregi cuprinse între 0 și 10000 inclusiv;
- pentru orice set de date de test care va fi folosit vor exista cel puțin K și cel mult 2.000.000.000 de cuvinte de lungime maximă care se pot forma respectând regulile descrise.

Exemplu

cuvinte.in

```
5 3
2 1 3 5 4
```

cuvinte.out

```
2 3 4
```

Explicație

Numerele din fișierul de ieșire corespund pozițiilor 2, 3 și 4 din șir pe care se află numerele 1, 3, respectiv 5.

Timp de execuție: 0,1 secunde/test

P080302 Timp

Marius Andrei

Avem la dispoziție o clepsidră mai ciudată cu care trebuie să măsurăm o anumită perioadă de timp.

Clepsidra este mai ciudată prin faptul că nu are doar două compartimente unde se poate strânge nisipul, ci trei.

Clepsidra se sprijină pe două dintre aceste compartimente, iar din al treilea se scurge nisipul în mod egal în celelalte. La început avem tot nisipul adunat într-un singur compartiment, iar cu el putem măsura, să presupunem 16 minute, ca în figura 1 a).

Figura 1

Pentru a măsura 7 minute în unul dintre compartimente putem roti clepsidra cu 120° în sensul acelor de ceasornic sau invers. Considerăm că rotația spre dreapta se efectuează în sensul acelor de ceasornic, iar rotația spre stânga se efectuează invers acelor de ceasornic. De exemplu, rotind spre dreapta obținem figura 1 b), în care ambele compartimente conțin nisip pentru a măsura 8 minute. Rotim spre stânga și obținem 4 și 12. Rotim spre dreapta și obținem 14 și 2, după care rotim spre dreapta și obținem 9 și 7. Astfel am obținut 7 într-un compartiment și putem măsura acest timp.

Scrieți un program care determină o serie de rotiri pentru a putea măsura un interval timp.

Date de intrare

Fișierul de intrare **timp.in** conține două numere întregi N și K separate între ele printr-un singur spațiu. N reprezintă timpul care poate fi măsurat inițial, tot nisipul aflându-se în stânga-jos (ca în exemplu), iar K reprezintă timpul pe care vrem să-l obținem într-un compartiment. Nu contează în ce parte se află nisipul cu care măsurăm timpul final.

Date de ieșire

În fișierul **timp.out** se va scrie pe prima linie numărul de rotiri necesare pentru a obține cantitatea cerută de nisip. Pe următoarele linii se va scrie câte o valoare care reprezintă rotirile în ordinea în care au fost efectuate. Pentru o rotație spre stânga se va scrie 0, iar pentru o rotație spre dreapta se va scrie 1.

Restricții și precizări

- $1 < K < N < 200.000.001$;
- soluția nu este unică și numărul de transformări nu trebuie să fie minim;
- după o rotație tot nisipul din partea de sus trebuie să se scurgă.

Exemplu

timp.in

```
16 7
```

timp.out

```
4
1
0
1
1
```

Timp de execuție: 0,1 secunde/test

P080303: Vecini

Marius Andrei

În *Drumul Taberei* există un bloc foarte ciudat. În primul an când a fost construit (să presupunem anul 1) avea un singur etaj, după care în fiecare an se construiește un nou etaj, astfel încât în anul X va avea X etaje.

Dar acesta nu este singurul lucru ciudat. Și modul în care este ocupat fiecare etaj este foarte ciudat.

La etajul 1 (primul etaj începând numărătoarea de jos) stă tot timpul administratorul, deci este ocupat. De asemenea ultimul etaj, fiind nou, este tot timpul ocupat. Restul etajelor însă sunt ocupate (sau libere) după regulile:

- dacă anul trecut etajul curent și etajul de dedesubt au fost ocupate, atunci etajul va fi liber anul acesta;

- dacă anul trecut etajul curent și etajul de dedesubt au fost libere, atunci etajul va fi liber și anul acesta;
- dacă anul trecut etajul curent a fost ocupat, iar cel de dedesubt a fost liber, atunci etajul curent va fi ocupat;
- dacă anul trecut etajul curent a fost liber, iar cel de dedesubt a fost ocupat, atunci etajul curent va fi ocupat.

Scrieți un program care determină configurația etajelor după N ani de la construcție.

Date de intrare

Fișierul de intrare **vecini.in** conține un singur număr natural N, care reprezintă numărul de ani.

Date de ieșire

În fișierul **vecini.out** se vor scrie N numere separate între ele printr-un spațiu. Numerele reprezintă starea fiecărui etaj, începând cu etajul 1. Reprezentarea este 0 pentru liber și 1 pentru ocupat.

Restricție

- $1 < N < 100.001$.

Exemplu

vecini.in

```
5
```

vecini.out

```
1 0 0 0 1
```

Timp de execuție: 0,2 secunde/test

P080304: Editor

Mugurel-Ionuț Andreica

Se consideră un editor de text care răspunde doar la apăsarea a șase taste, și anume cele care au tipărite simbolurile: "(", ")", "[", "]", "*" și "E". Dacă se apasă una din tastele "(", ")", "[", "]", atunci se afișează caracterul respectiv pe ecran. Dacă se apasă tasta "*" se șterge ultimul caracter afișat (dacă nu este afișat nici un caracter, atunci nu se întâmplă nimic). Dacă se apasă tasta "E" (*Enter*), atunci editorul va verifica dacă șirul afișat pe ecran este un șir parantezat corect. Un șir este parantezat corect dacă este construit conform regulilor:

- *<șir parantezat corect>* = *<șirul vid>*;
- *<șir parantezat corect>* = "(" + *<șir parantezat corect>* + ")";
- *<șir parantezat corect>* = "[" + *<șir parantezat corect>* + "]";
- *<șir parantezat corect>* = *<șir parantezat corect 1>* + *<șir parantezat corect 2>*.

Prin $X + Y$ s-a notat concatenarea șirurilor X și Y.

De exemplu, șirurile "[]([([])])", "[[[([[[]])]]](())" și "([([])[()]])[]" sunt parantezate corect, iar șirurile ")[]([])", "[()()()()[([(" și "[(()[([]())]])" nu sunt parantezate corect.

Dându-se o succesiune de taste apăsate care se termină cu tasta "E" și știind că, inițial, nu este nici un caracter afișat pe ecran, trebuie să decideți dacă șirul afișat pe ecran în urma apăsării tastelor este un șir parantezat corect.

Date de intrare

Pe prima linie a fișierului de intrare **editor.in** se află un număr întreg T, reprezentând numărul de succesiuni de taste care vor fi descrise în continuare. Pe fiecare din următoarele T linii este descrisă câte o succesiune de taste apăsate.

Date de ieșire

În fișierul **editor.out** veți scrie exact T linii, câte una pentru fiecare succesiune de taste descrisă în fișierul de intrare.

Pe fiecare linie veți afișa ":)" (fără ghilimele) dacă șirul obținut în urma succesiunii corespunzătoare de taste apăsate este parantezat corect , respectiv ":(" (fără ghilimele) în caz contrar.

Restricții și precizări

- $1 \le T \le 30$;
- nici o succesiune de caractere nu va conține mai mult de 60000 de taste apăsate;
- nu se acordă punctaje parțiale.

Exemplu

editor.in

```
6
E
)))))*****]]]]]*****E
(([[()[()[]([])][[]()([])](())]]))([
][][])E
**[*]*[]()(]*(**)*E
([](()))E
[E
```

editor.out

```
:)
:)
:)
:)
:(
:(
```

Timp de execuție: 1 secundă/test

P080305: Numere

Bogdan Stroe

Ionel are o slăbiciune pentru jocurile de perspicacitate. El încearcă să rezolve un astfel de joc în care are o matrice de dimensiune $M \times N$ ale cărei elemente sunt numere întregi.

O mutare constă în scăderea dintr-un element a numărului de vecini ai poziției elementului respectiv și creșterea cu unu a valorilor din pozițiile vecine.

Vecinii pentru o anumită poziție sunt elementele de pe orizontală și verticală, deci o poziție poate avea 2, 3 sau 4

vecini, după cum este situată într-un colț, pe o latură sau în interiorul matricei.

De exemplu, având matricea cu două linii și trei coloane:

```
1 2 4
4 2 5
```

o mutare la linia 1 și coloana 2 ar duce la matricea:

```
2 -1 5
4  3 5
```

Anumite poziții cunoscute din matrice sunt *sensibile la scădere*, adică nu se pot efectua mutări în pozițiile respective. Numărul acestor poziții este egal cu *K*.

Scopul lui *Ionel* este ca, pornind de la matricea inițială, să obțină o matrice în care toate elementele să aibă aceeași paritate, efectuând un număr minim de mutări.

Date de intrare

Pe prima linie a fișierului de intrare **numere.in** se află trei numere întregi *M*, *N* și *K*, cu semnificațiile din enunț.

Pe următoarele *M* linii se află câte *N* numere întregi cuprinse între 1 și 10000 inclusiv, reprezentând matricea inițială.

Pe următoarele *K* linii se află câte două numere întregi separate prin spațiu, reprezentând coordonatele pozițiilor *sensibile la scădere* (prima coordonată corespunde liniei, iar cea de-a doua coloanei).

Date de ieșire

În fișierul de ieșire **numere.out** se va afișa pe prima linie numărul minim de mutări *MIN* care rezolvă jocul.

Pe următoarele *MIN* linii se vor afișa mutările, câte o mutare pe linie (două numere întregi separate printr-un spațiu, reprezentând linia și coloana poziției fiecărei mutări).

În cazul în care nu există soluție, în fișierul de ieșire se va scrie, pe un singur rând, mesajul: `NU EXISTA SOLUTIE`.

Restricții și precizări

- $2 \le M, N \le 12$;
- $0 \le K \le M \cdot N$;
- elementele din matrice sunt numere întregi cuprinse între 1 și 10000 inclusiv; pe parcursul jocului, este permis ca elementele din matrice să aibă valori în afara acestui interval;
- dacă există mai multe soluții cu număr minim de mutări, atunci se va afișa una dintre ele.

Exemplu

numere.in

```
3 3 1
0 3 0
2 4 -4
9 8 3
3 1
```

numere.out

```
2
1 2
2 2
```

Timp de execuție: 1 secundă/test

P080306: Oo

Osman Ay

Fermierul *Ion* are o fermă de formă circulară, unde cresc *N* găini. Ferma a fost împărțită în *N* sectoare, numerotate de la 1 la *N*, astfel încât oricare două sectoare având numere consecutive sunt adiacente (se află unul lângă altul). În plus, primul și ultimul sector sunt adiacente. În fiecare sector se află câte o găină, iar aceasta depune un anumit număr de ouă în fiecare zi.

După ce găinile depun ouăle, fermierul *Ion* dorește să le adune, pentru a le mânca. Deoarece fermierul este foarte lacom, de fiecare dată el alege două sectoare adiacente din care adună ouale simultan.

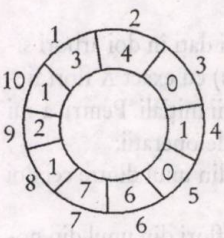


Figura 2: Ferma

Din păcate, din cauza lăcomiei sale, găinile din sectoarele vecine cu cele două alese se sperie și devin violente, motiv pentru care fermierul nu mai poate aduna ouăle din aceste sectoare. În exemplul din figură, dacă fermierul adună ouăle din sectoarele 1 și 2, el nu va mai putea aduna ouăle din sectoarele 3 și 10.

Determinați numărul maxim de ouă pe care le poate aduna fermierul *Ion*, în urma aplicării strategiei sale lacome.

Date de intrare

Fișierul de intrare **oo.in** conține pe prima linie numărul de sectoare în care este împărțită ferma (și, implicit, numărul de găini). Pe următoarea linie se află *N* numere întregi, reprezentând numărul de ouă depuse de fiecare găină, în ordinea sectoarelor în care se află acestea.

Date de ieșire

În fișierul de ieșire **oo.out** se va afișa un număr întreg care reprezintă numărul maxim de ouă pe care le poate aduna fermierul *Ion*.

Restricții și precizări

- $2 \le N \le 10.000$;
- numărul de ouă pe care le poate depune o găină în fiecare zi este un număr întreg cuprins între 0 și 100 inclusiv.

Exemplu

oo.in

```
10
3 4 0 1 0 6 7 1 2 1
```

oo.out

```
20
```

Explicație
Fermierul poate aduna ouăle din sectoarele 2 și 3 (4+0), 6 și 7 (6+7) și 9 și 10 (2+1).

Timp de execuție: 0,1 secunde/test

Clasele a XI-a și a XII-a

P080307: Arbori

Mihai Stroe

Am doi copaci reprezentați prin doi arbori binari cu M, respectiv N noduri. În fiecare nod al unui astfel de arbore cresc flori. Ordinea ramurilor (fiilor) contează (se face distincție între fiul stâng și fiul drept).

Doi arbori binari cu flori în noduri sunt similari dacă (definiție recursivă):
- au ambii 0 noduri, sau
- au ambii cel puțin un nod, același număr de flori în rădăcină, subarborii stângi sunt similari și subarborii drepți sunt similari.

Vreau să transform cei doi arbori dați în doi arbori similari (conform definiției de mai sus) cu exact K flori fiecare, având aceleași rădăcini cu arborii inițiali. Pentru a-mi atinge scopul pot să fac două tipuri de operații:
- tai o ramură (elimin un subarbore din unul dintre cei doi arbori);
- rup o floare (scad cu 1 numărul de flori din unul din nodurile unuia dintre arbori).

Deoarece vreau să muncesc cât mai putin pentru a-mi atinge scopul, doresc să tai cât mai puține ramuri și, dacă am mai multe variante de a tăia cât mai puține ramuri, să aleg una în care să rup cât mai puține flori.

Scrieți un program care să mă ajute.

Date de intrare
Fișierul de intrare **arbori.in** conține pe prima linie numărul K de flori care trebuie să rămână în cei doi arbori. Urmează cei doi arbori, unul după altul.

Pentru fiecare arbore, pe prima linie se află numărul de noduri NR. Urmează NR linii, fiecare conținând informația pentru un nod: numărul de flori F, numărul fiului stâng (sau 0 dacă acesta nu există) și numărul fiului drept (sau 0 dacă acesta nu există).

Arborii sunt valizi (conțin numerele de la 1 la NR, nu au cicluri etc.) și au amândoi rădăcina în nodul 1.

Date de ieșire
În fișierul **arbori.out** se va afișa pe prima linie numărul T de tăieri de ramuri și numărul R de ruperi de flori din soluția optimă. Pe următoarele T linii veți afișa tăierile de ramuri. Tăierea unei ramuri se reprezintă prin două numere ARB și NOD; aceasta înseamnă că am tăiat ramura din arborele ARB (1 sau 2), care lega nodul NOD de tatăl lui. Pe următoarele R linii veți afișa ruperile de flori. Ruperea unei flori se reprezintă prin două numere ARB și NOD; aceasta înseamnă că am rupt o floare din nodul NOD al arborelui ARB. Numerele de pe fiecare linie se vor separa printr-un spațiu. Dacă există mai multe soluții optime, afișați una oarecare. Toate testele date vor avea cel puțin o solutie.

Restricții
- $1 \le M, N \le 100$;
- $0 \le F \le 10$;
- $0 \le K \le 100$;
- $1 \le NR \le 100$.

Exemplu

arbori.in
```
7
3
5 2 3
2 0 0
1 0 0
3
6 2 0
6 0 3
5 0 0
```

arbori.out
```
2 5
1 3
2 3
2 1
2 2
2 2
2 2
2 2
```

Timp de execuție: 1 secundă/test

P080308: Sum

Marius Andrei

Se dă un șir de numere întregi. Se caută un subșir cu lungimea cuprinsă între L și U, format din elemente consecutive ale șirului inițial, cu suma elementelor maximă.

Scrieți un program care determină suma maximă a unui astfel de subșir.

Date de intrare
Fișierul de intrare **sum.in** conține pe prima linie trei numere întregi N, L și U, despărțite între ele printr-un spațiu. N reprezintă lungimea șirului mare. L este lungimea minimă și U este lungimea maximă. Pe următoarele linii se află cele N numere, despărțite prin câte un spațiu.

Date de ieșire
În fișierul **sum.out** se va scrie pe prima linie suma maximă care se poate obține.

Restricții și precizări

- $1 \le L \le U \le N \le 100.001$;
- numerele din șir sunt numere întregi și au valori din intervalul [-10 000, 10 000].

Exemplu

sum.in

```
9 2 3
100 -100 0 10 -5 0 10 0 1
```

sum.out

```
11
```

Timp de execuție: 0,1 secunde/test

P080309: Taxi

Marius Andrei

Orașul *X*3 a fost gândit de informaticieni, astfel încât străzile să împartă orașul sub forma unei matrice. Intersecțiile dintre străzi sunt practic punctele cu coordonate întregi cuprinse în intervalul [0, *A*] pentru coordonata *X* și [0, *B*] pentru coordonata *Y*.

În acest oraș există exact două taxiuri. Pentru că s-au înțeles între ei, ambii taximetriști percep același tarif pentru aceeași distanță parcursă. Așa că și oamenilor le este indiferent cu care dintre taxiuri se deplasează. Când este nevoie de taxi, singurul criteriu este taxiul care este mai aproape. Cetățenii sunt foarte bine informați și cunosc la ce coordonate se află taxiurile. Nu le rămâne decât să decidă care este mai aproape. Probleme serioase apar atunci când taxiurile sunt egal depărtate pentru că cetățenii nu se pot hotărî și rămân pe loc ore întregi.

Azi fiind o zi foarte aglomerată, în fiecare intersecție se află exact un om care dorește să apeleze la un taxi.

Scrieți un program care determină câți oameni vor rămâne nehotărâți.

Date de intrare

Fișierul de intrare **taxi.in** conține pe prima linie numărul de teste *T*.

Pe următoarele *T* linii se află câte 6 numere întregi, separate printr-un spațiu reprezentând *A* (coordonata maximă pentru *X*), *B* (coordonata maximă pentru *Y*) și coordonatele celor două taxiuri, și anume *x*1 *y*1 *x*2 *y*2.

Date de ieșire

În fișierul **taxi.out** se vor scrie *T* linii, câte una pentru fiecare test. Pe fiecare linie se va scrie un număr întreg, reprezentând numărul de persoane indecise pentru testul respectiv.

Restricții și precizări

- $0 < T < 101$;
- $0 < A, B < 10.001$;
- $0 \le x_1, x_2 \le A$ și $0 \le y_1, y_2 \le B$.

Exemplu

taxi.in

```
1
9 9999 0 0 2 0
```

taxi.out

```
10000
```

Timp de execuție: 0,1 secunde/test

P080310: Expr

Radu Berinde

Acarie, un student eminent, are probleme cu tema pentru seminarul de structuri algebrice. A redus problema la una mai simplă, care constă în operații cu mulțimi. Însă sunt prea multe operații și deja este timpul să plece în oraș. Ajutați-l să iasă din încurcătură.

Scrieți un program care găsește rezultatul unei expresii valide date.

Date de intrare

În fișierul de intrare **expr.in** se află un șir de caractere fără spații. Acest șir de caractere poate conține următoarele elemente:

- operanzii - sunt mulțimi, descrise de o acoladă "`{`" urmată de numerele care fac parte din mulțimea respectivă (în ordine crescătoare) despărțite de "`,`" (virgulă) și terminate cu o acoladă "`}`";
- operatorii - care pot fi paranteze, sau una din următoarele operații (care au aceeași prioritate):
  - `*` - intersecție;
  - `+` - reuniune;
  - `-` - diferență
  - `%` - diferență simetrică: `A % B = (A - B) + (B - A) = (A + B) - (A * B)`.

Date de ieșire

Scrieți rezultatul expresiei în fișierul **expr.out** ca o singură mulțime (după formatul mulțimilor din fișierul de intrare).

Restricții și precizări

- lungimea șirului din fișierul de intrare este mai mică sau egală cu 100.000;
- numărul valorilor distincte care apar în expresie este mai mic sau egal cu 8.000;
- valorile care apar în mulțimi sunt numere întregi din intervalul [0, 2.000.000.000];
- adâncimea maximă de parantezare nu depășește valoarea 100;
- numărul maxim de operații care apar în expresie este mai mic sau egal cu 10.000;
- mulțimea vidă este reprezentată prin "`{}`";
- numerele din mulțimea rezultată vor fi afișate în ordine crescătoare.

Exemplu

expr.in

```
{1,2,3,4}%({1,2,3,4}*{}+{1,2}+{5,6}-{1})
```

expr.out

```
{1,3,4,5,6}
```

Timp de execuție: 0,2 secunde/test

P080311: Ciclu

Radu Berinde

Acarie, fericit că a trecut examenul la algebră pleacă în oraș să sărbătorească. Este atât de fericit, încât nu se mulțumește cu un simplu traseu între terase, ci vrea un ciclu pentru a fi sigur că poate continua să petreacă atâta timp cât dorește. Din păcate, drumurile dintre terase au costuri diferite și *NU* sunt bidirecționale. *Acarie* nu este interesat neapărat să minimizeze costul acestui ciclu, ci dorește să minimizeze costul *MEDIU* al ciclului (evident, vrea să meargă cât mai puțin între terase, astfel încât costul ciclului împărțit la numărul de drumuri dintre terase să fie minim).

Ajutați-l și de această dată pe *Acarie* și vă va cinsti toată ziua. Scrieți un program care determină costul mediu minim al unui ciclu de terase.

Date de intrare

În fișierul de intrare **ciclu.in** se află N (numărul de terase), M (numărul de drumuri unidirecționale) și cele M drumuri directe dintre terase. Acestea sunt scrise fiecare pe câte o linie, reprezentate de trei numere întregi - terasa de unde pornește drumul, terasa unde ajunge drumul (numere între 1 și N) și costul drumului (întreg strict pozitiv mai mic decât 100000).

Date de ieșire

Să se afișeze în fișierul **ciclu.out** costul mediu minim al ciclului găsit, cu două zecimale exacte.

Restricții

- $2 \le N \le 1000$;
- $2 \le M \le 4000$;
- există cel puțin un ciclu.

Exemplu

ciclu.in

```
4 5
1 2 1
2 3 1
1 3 1
3 4 2
4 1 3
```

ciclu.out

```
1.75
```

Timp de execuție: 3 secunde/test

P080312: Prefix

Mugurel-Ionuț Andreica

Se consideră un șir format din literele mici 'a', 'b', ...,'z'. Să se determine cel mai lung prefix periodic al său. Un șir X este periodic dacă se poate scrie sub forma $P + P + ... + P$, unde prin $A + B$ s-a notat concatenarea șirurilor A și B. Șirul P se numește perioada lui X și trebuie să fie strict mai scurt decât X.

De exemplu, șirurile "abcaabcaabcaabca", "xyyxyy" și "wwwww" sunt periodice, iar șirurile "abcaabcaabcaz", "xyxyxz" și "wwwaawww" nu sunt periodice.

Date de intrare

Prima linie a fișierului de intrare **prefix.in** conține numărul întreg T de șiruri prezente în fișier. Fiecare din următoarele T linii conține câte un șir.

Date de ieșire

În fișierul de ieșire **prefix.out** se va afișa, pentru fiecare din cele T șiruri din fișierul de intrare, lungimea celui mai lung prefix periodic.

Restricții și precizări

- $1 \le T \le 10$;
- fiecare șir va avea cel puțin unul și cel mult 1.000.000 de caractere;
- nu se acordă punctaje parțiale.

Exemplu

prefix.in

```
10
abcdefgh
z
xxxuxxxu
abbcaabbcaabbcaabbcaxyzxyzxyzxyz
hellohellohellohellauhellohello
aaaaaaaaazaaaaaaaaazaaaaaaaaa
uvwuvwuvwuvwu
sirperiodicsirperiodicsirperiodicsir
aababcabcdabcdeabcdefabcdefgaerror
mamatatabunicubunicaunchiumatusa
```

prefix.out

```
0
0
8
20
15
20
12
33
2
4
```

Timp de execuție: 1,5 secunde/test

ACM

Mihai Stroe

Vă prezentăm în continuare soluțiile problemelor propuse spre rezolvare la concursul internațional de programare ACM, ediția 2003/2004 care a avut loc la București. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de către autorii problemelor.

P070301: Codificare delta

Aceasta este o problemă "muncitorească". Dificultatea ei constă în implementarea rapidă, în timpul concursului, a algoritmului (specificat în enunț) și respectarea restricțiilor impuse asupra datelor de intrare și ieșire.

Algoritmul de decriptare se deduce în mod evident din algoritmul de criptare, folosind aritmetica modulo 26. Testarea validității cifrului este foarte simplă.

Analiza complexității

Toate operațiile se pot efectua în timp proporțional cu lungimea datelor asupra cărora se execută.

Ordinul de complexitate a algoritmului este $O(L)$, unde L este lungimea datelor de intrare.

P070302: Parola ascunsă

În continuare vom descrie algoritmul de rezolvare a acestei probleme pentru un test, urmând ca acesta să se aplice de câte ori este nevoie.

Mai întâi se dublează șirul inițial prin concatenare cu el însuși.

Soluția problemei este dată de către un indice X cuprins între 1 și L inclusiv, pentru care subșirul de L caractere al șirului dublat, care începe din poziția X, este primul în ordine lexicografică.

Vom nota cu S_i subșirul de N caractere care începe din poziția i.

Pentru testele date, folosirea funcției `strncmp` ar fi dus la o rezolvare eficientă. Aceasta poate compara două subșiruri de L caractere, terminându-se în momentul în care se întâlnește o diferență.

Ordinul de complexitate al acestui algoritm este $O(L^2)$, deci teoretic această variantă nu s-ar fi încadrat în timp (concurenții nu știau numărul de teste etc.). Practic, pe testele date în concurs și eventual folosind câteva euristici suplimentare, această rezolvare ar fi fost acceptată.

Concurentii mai experimentați au preferat rezolvări mai eficiente. Astfel, folosind structuri de date avansate (arbori de sufixe, vectori de sufixe), se pot obține algoritmi de complexitate $O(L \cdot \log^2 L)$ sau $O(L \cdot \log L)$.

Să examinăm o astfel de rezolvare.

Se vor executa $O(\log L)$ pași. După fiecare pas P se va obține o listă de grupe de indici, ordonată în funcție de ordinea șirurilor de lungime 2^P care încep din pozițiile determinate de indicii respectivi. Dacă avem subșiruri identice de lungime 2^P, acestea vor face parte din aceeași grupă. P crește până când o nouă creștere ar duce la depășirea lui L.

Presupunând că putem obține această structură după P pași, să vedem cum o construim pe cea de la pasul $P + 1$.

Pentru a construi structura de la pasul $P + 1$ trebuie să putem compara șiruri care încep cu indici i și j aflați în aceeași grupă la pasul P. Comparația este simplă dacă ținem cont de faptul că un subșir de lungime 2^{P+1} este format din doua jumătăți de lungime 2^P. Deoarece în cazul indicilor aflați în aceeași grupă primele jumătăți sunt egale, trebuie comparată jumătatea a doua a celor două șiruri; pentru aceasta se determină în ce grupă se aflau jumătățile finale, deci indicii $2^P + i$ și $2^P + j$. Această informație poate fi construită pe parcurs, deci comparația are ordinul de complexitate $O(1)$.

Cunoscând aceste detalii, la pasul $P+1$ putem sorta elementele din fiecare grupă de la pasul P, folosind un algoritm de complexitate $O(NR \cdot \log NR)$, unde NR este numărul de elemente din grupă (maxim L).

Pentru aflarea rezultatului se vor testa indicii din prima grupă. Aceștia determină șiruri egale de lungime 2^P. Pentru a compara șiruri de lungime N se testează ultimele 2^P caractere din aceste șiruri; aceasta se realizează folosind indici de forma $i + (L - 2^P)$.

Analiza complexității

În continuare vom calcula ordinul de complexitate al algoritmului eficient prezentat.

Avem în total $P = \log L$ pași, iar la fiecare pas se realizează câte o sortare în fiecare grupă. Cazul cel mai defavorabil se obține când una din grupe este mare în comparație cu celelalte. în acest caz, ordinul de complexitate al algoritmului este $O(L \cdot \log L)$.

În concluzie, ordinul total de complexitate pentru fiecare set de date este $O(L \cdot \log^2 L)$.

P070303: Petrecerea nebunatică

În continuare vom rezolva această problemă pentru un singur test, urmând ca soluția să fie aplicată pentru fiecare test din fișierul de intrare.

Cunoaștem că n este numărul de participanți la petrecere și considerăm permutarea identică cu n elemente (1, 2, ..., n).

Împărțim această permutare în două permutări, astfel încât fiecare dintre ele să aibă $[n/2]$, respectiv $n - [n/2]$ elemente.

Dacă n este număr par, adică $n = 2 \cdot k$, atunci pentru a inversa pe fiecare dintre cele două jumătăți avem nevoie de $k \cdot (k - 1)/2$ operații. În total se efectuează $k \cdot (k - 1)$ operații, adică $(n^2 - 2 \cdot n)/4$ operații.

Dacă n este număr impar, adică $n = 2 \cdot k + 1$, atunci pentru a inversa o jumătate din permutare se efectuează $k \cdot (k - 1)/2$ operații, iar pentru cealaltă $k \cdot (k + 1)/2$ operații. Rezultă că, în total, se efectuează $(n^2 - 2 \cdot n + 1)/4$ operații.

În timpul concursului trebuia ținut cont de faptul că rezolvarea se putea trimite de mai multe ori. Se trimitea o rezolvare de tipul celei prezentate anterior și se obținea răspunsul *Accepted* (*Soluție acceptată*), fără a se mai verifica dacă existau soluții mai bune. Nu existau.

Analiza complexității

Pentru un singur test ordinul de complexitate a rezolvării este $O(1)$ deoarece ordinul de complexitate a operațiilor de citire, scriere și calculare a timpului necesar pentru ca participanții să fie aranjați în ordine inversă este $O(1)$.

Dacă luăm în considerare toate testele și notăm cu t numărul acestora, ordinul de complexitate devine $O(t)$.

P070304: Supermagazin

În continuare vom prezenta soluția acestei probleme pentru un singur test.

Vom încerca să construim soluția optimă în ordine "cronologică invers".

Fie T momentul de timp maxim la care se poate vinde un produs.

În ultima unitate de timp pot fi vândute doar produse cu timpul limită de vânzare T. Este bine să profităm de această unitate de timp, deci alegem unul din acele produse și îl vindem. Mai exact, alegem produsul pe care obținem profit maxim.

La momentul $T - 1$ poate fi vândut fie un produs care are timpul limită de vânzare $T - 1$, fie unul din produsele care aveau timpul limită de vânzare T și care nu a fost vândut. Se alege produsul pe care se obține profit maxim.

La momentul de timp $K < T$ putem vinde orice produs cu timpul limită de vânzare mai mare sau egal cu K și care nu a fost deja vândut. Se alege produsul pe care se obține profit maxim.

Un set de date are n produse, maxim 10000. Dacă la fiecare pas alegem minimul cu metoda clasică, complexitatea este $O(N^2)$. Pentru un set de date cu multe teste mari, teoretic acest algoritm nu s-ar încadra în timp. Soluția este să folosim un *heap* pentru extragerea maximului. La fiecare moment de timp K (K descrește pornind de la T) se introduc în *heap* profiturile corespunzătoare produselor cu timpul limită de vânzare K, apoi se extrage maximul și se adună la soluție.

Analiza complexității

Operația de citire a datelor de intrare pentru un singur test are ordinul de complexitate $O(n)$, iar cea de scriere a rezultatului are ordinul de complexitate $O(1)$.

Calcularea profitului planificării optime are ordinul de complexitate $O(n \cdot \log n)$, deoarece fiecare dintre cele n produse este introdus în *heap*, iar ordinul de complexitate de introducere și extragere de elemente din acesta este $O(\log n)$.

Dacă m este numărul de teste dintr-un set de date, atunci ordinul de complexitate al algoritmului de rezolvare pentru această problemă devine $O(m \cdot n \cdot \log n)$.

P070305: Stăpânul inelului

Problema se rezolvă prin metoda *backtracking*. Ținând cont de testele date în concurs, limitele din enunț sunt dimensionate mai sus decât ar trebui. Astfel, testul maxim a constat în 200 de tufe, situate la distanța 1 unele de altele. Celelalte teste aveau mai puțin de 10 tufe, distanța între cele mai depărtate două fiind cel mult 50. Pe nici un test produsul cerut nu a depășit 50000. O dimensionare a limitelor corespunzătoare cu testele date ar fi încurajat mai mulți concurenți să abordeze problema.

Pentru a calcula produsul distanțelor dintre tufele consecutive se vor determina pozițiile tufelor. Prima tufă va avea coordonata 0, iar a doua coordonata *MAX*, unde *MAX* este maximul distanțelor.

După citirea datelor se creează doi vectori: unul care conține distanțele distincte, altul care conține numărul de apariții al fiecărei distanțe.

Abordarea *backtracking* constă în alegerea, la fiecare pas, a poziției în care va fi plasată o tufă. Mai exact, după ce le-am plasat pe primele K (prima la coordonata 0, a doua la coordonata *MAX*), tufa $K + 1$ va fi plasată la o poziție între coordonata tufei K și *MAX*. Făcând abstracție de tufa 2, coordonatele se vor obține în ordine crescătoare.

Pentru fiecare variantă de plasare se va verifica dacă există distanțele între tufa $K + 1$ și primele K tufe în vectorul de distanțe, folosind căutarea binară, și dacă numărul de apariții este strict pozitiv. În caz afirmativ, se scade numărul de apariții al distanțelor respective (deci considerăm distanțele ca fiind alocate) și se încearcă plasarea celorlaltor tufe. Numărul de apariții va fi restaurat la abandonarea variantei curente.

Limitarea produsului ne asigură că, dacă există o soluție, majoritatea distanțelor între tufe consecutive vor fi egale cu 1. Altfel spus, de multe ori *backtracking*-ul va înainta rapid, plasând mai multe tufe consecutive, fiecare la distanța 1 față de precedenta.

Dacă pe parcursul generării soluției, produsul distanțelor dintre tufele consecutive deja plasate depășește limita din enunț, varianta curentă nu este corectă. De asemenea, dacă distanța dintre tufa curentă și ultima tufă este mai mică decât numărul de tufe care încă nu au fost plasate, varianta curentă este abandonată.

Analiza complexității

Algoritmul de rezolvare a acestei probleme are ordinul de complexitate exponențial. Un calcul mai exact ar trebui să țină cont de limitele din enunț sau de limitele din testele date (cazuri diferite). Lăsăm efectuarea acestui calcul ca exercițiu pentru cititor.

În orice caz, un program care implementează această rezolvare rulează instantaneu pe testele date.

P070306: Subsecvența comună

Problema cere calcularea celei mai lungi subsecvențe comune a două șiruri de caractere. Aceasta este una din primele probleme care se învață la *metoda programării dinamice*.

În continuare prezentăm modul de rezolvare, pentru un singur test, bazat pe metoda *programării dinamice*.

Notăm șirurile cu X și Y și cu M și N lungimile acestora. Construim o matrice A cu M linii și N coloane, în care $A_{i,j}$ reprezintă lungimea celei mai lungi subsecvențe comune care se poate obține folosind doar primele i litere din X și primele j litere din Y.

În continuare prezentăm modul de construire al elementelor matricei A.

În cazul în care $X_i = Y_j$, atunci $A_{i,j} = A_{i-1,j-1} + 1$, deoarece subsecvența respectivă va fi formată din subsecvența comună a șirurilor, folosind primele $i - 1$, respectiv $j - 1$ litere, la care se va adăuga litera X_i.

În caz contrar, se alege maximul dintre $A_{i-1,j}$ și $A_{i,j-1}$, pentru că una din literele finale nu va intra în subsecvență.

Soluția problemei este dată, în final, de valoarea elementului $A_{M,N}$.

Analiza complexității

Operația de citire a datelor pentru un test are ordinul de complexitate $O(M + N)$, iar operația de afișare a rezultatelor are ordinul de complexitate $O(1)$.

Calculul unui element al matricei A se realizează în timp constant. Ordinul de complexitate al construirii matricei A este $O(M \cdot N)$, unde M și N sunt lungimile celor două șiruri.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(T \cdot M \cdot N)$, unde T reprezintă numărul de teste.

P070307: Rețeaua electrică

Rețeaua electrică este un graf orientat, care se transformă într-o rețea de transport astfel:

- capacitatea unui arc reprezintă transportul maxim pe arcul respectiv;
- se introduce o sursă fictivă din care pleacă spre toți producătorii de curent arce de capacitate $+\infty$;
- se introduce o destinație fictivă în care sosesc arce de capacitate $+\infty$ de la toți consumatorii.

Pe această rețea de transport se aplică un algoritm de flux maxim.

În final, soluția problemei este dată de suma fluxului pe arcele care pleacă din sursa fictivă.

Subiectul fluxului într-o rețea de transport este mult prea amplu pentru a fi tratat în cadrul acestui articol.

În continuare vom examina algoritmul general *Ford-Fulkerson*, împreună cu câteva optimizări. Amintim și existența algoritmilor de preflux, ceva mai dificil de înțeles, dar mai eficienți. Sugerăm cititorului interesat să studieze capitolul despre flux din cartea *Introducere în algoritmi, T. H. Cormen, C. E. Leisserson, R. R. Rivest.*

Pe scurt, fluxul pe arcele rețelei este dat de valorile $l(i, j)$ transportate de fiecare arc, iar capacitatea arcelor este dată de $L_{max}(i, j)$.

Fie un flux valid prin rețea. Acesta trebuie să respecte condițiile de conservare (ceea ce intră într-un nod intermediar este egal cu ceea ce iese) și mărginire (fluxul pe orice arc este cel mult egal cu capacitatea arcului respectiv).

Evident, fluxul 0 pe toate arcele este un flux valid.

Algoritmul *Ford-Fulkerson* este următorul:

- se pornește de la fluxul inițial 0 pe toate arcele;
- cât timp acest flux poate fi mărit (există un drum de creștere de la sursă la destinație), se modifică fluxul pe arcele drumului de creștere, obținându-se un flux mai bun.

Un drum de creștere este un drum de la sursă la destinație, folosind:

- arce (i, j) pe care fluxul este mai mic decât capacitatea (deci mai poate fi crescut);
- arce fictive (i, j) pentru care fluxul pe arcul (j, i) este nenul (deci poate fi scăzut). Practic folosirea unor arce din această categorie conduce la redirecționarea fluxului prin rețea.

Determinarea existenței unui drum de creștere de la sursă la destinație are ordinul de complexitate $O(m)$, printr-o parcurgere a grafului care contine arcele de mai sus. Pentru a calcula complexitatea totală a algoritmului este necesar să estimăm câte drumuri de creștere vor fi găsite până la determinarea fluxului maxim.

Dacă la fiecare pas se alege un drum oarecare, complexitatea pe cazul cel mai defavorabil este $O(m \cdot F_{max})$, unde F_{max} este valoarea fluxului maxim. Această limită se atinge în cazul în care fluxul în retea crește cu 1 la fiecare pas. Evident, pentru grafuri cu capacități mari pe muchii (deci unde fluxul maxim poate avea valori mari) acest algoritm nu este acceptabil.

O variantă este algoritmul *Edmonds-Karp*. Acest algoritm implementează căutarea drumului de creștere printr-o

parcurgere în lățime, obținându-se la fiecare pas un drum de creștere cu un număr minim de arce. Se poate demonstra că această variantă are ordinul de complexitate $O(n \cdot m^2)$.

O altă variantă este dată de algoritmul de flux maxim prin scalare. Astfel, fie C capacitatea maximă a unei muchii, $P = [\log C]$ și $K = 2^P$.

Primele drumuri de creștere căutate trebuie să aibă capacitatea cel puțin K, adică arcele pe care se introduce flux să permită introducerea a cel putin K unități, iar arcele de pe care se "scoate" (se redirecționează pe alte arce) flux să aibă fluxul cel puțin egal cu K. În momentul în care nu mai există astfel de drumuri se vor căuta drumuri de capacitate $K/2$, apoi $K/4$ etc.

Prin această metodă, se introduce o cantitate importanta în rețea încă de la primele iterații.

Se poate demonstra că acest algoritm are ordinul de complexitate $O(m^{2 \cdot \log C})$.

În practică, toți algoritmii de mai sus au timpi de execuție mult mai mici decât indică ordinul de complexitate. Astfel, la căutarea unui drum de creștere nu se examinează întotdeauna toate cele m muchii.

Pentru valori ale lui m apropiate de n (de exemplu, cazul grafurilor planare), metoda prin scalare poate conduce la performanțe excelente, ordinul de complexitate devenind $O(n^{2 \cdot \log C})$ față de $O(n^3)$ la algoritmii de preflux.

Analiza complexității

Operația de citire a datelor (mai dificilă pentru programatorii *Pascal* datorită parantezelor și virgulelor) are ordinul de complexitate $O(m + n)$.

Operația de transformare a grafului într-o rețea de transport are ordinul de complexitate $O(n)$.

Datorită faptului că ordinul de complexitate al oricărui algoritm de determinare a fluxului maxim într-o rețea de transport este mai mare decât celelalte operații suplimentare, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este dat de ordinul de complexitate al algoritmului de flux maxim folosit pe structura cu ajutorul căreia s-a reținut graful.

P070308: Pompierii

Această problemă se poate rezolva analizând toate posibilitățile de a înlocui semnele de întrebare cu operatori și evaluând de fiecare dată expresia rezultată. Pentru 10 semne de întrebare se obțin $4^{10} = 2^{20} = 1048576$ variante; aparent sunt destul de multe, dar calculatoarele din concurs aveau procesoare cu frecvența 2,6 *GHz*.

În momentul în care s-a obținut o expresie care dă rezultatul corect se întrerupe generarea posibilităților pentru testul respectiv și se afișează `yes`. Dacă nu se obține nici o astfel de expresie, se afișează `no`.

Pentru a optimiza pasul de evaluare a expresiei se pot determina poziția semnelor, a parantezelor, valorile operanzilor inițiali și ordinea aproximativă a operațiilor înainte de generarea tuturor variantelor. Mai exact, ordinea operațiilor dintre paranteze (dacă acestea există) este fixată datorită precizărilor din enunț.

Analiza complexității

Pentru un singur test, operația de citire a datelor de intrare are ordinul de complexitate $O(L)$, unde L este lungimea expresiei, iar operația de scriere a rezultatelor are ordinul de complexitate $O(1)$.

Fie Op numărul de operatori și S numărul de semne de întrebare.

Ordinul de complexitate al operației de generare a tuturor expresiilor posibile este $O(4^S)$.

Pentru fiecare expresie, evaluarea ei are ordinul de complexitate $O(Op)$.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N \cdot 4^S)$ pentru fiecare set de date.

Dacă luăm în considerare toate testele din setul de date, atunci ordinul de complexitate al algoritmului de rezolvare pentru această problemă devine $O(T \cdot N \cdot 4^S)$, unde T este numărul total de teste.

P070309: Polinoame binare

Problema se rezolvă prin metoda forței brute.

Se generează toți cei $C(n, k)$ vectori binari. Pentru fiecare din acești vectori se evaluează funcția dată. Se afișează numărul de vectori pentru care rezultatul este 1.

Pentru generare se poate folosi fie metoda *backtracking*, fie o simplă numărare (adică un ciclu cu 2^n pași). În cazul numărării, deși se parcurg toți cei 2^n vectori cu n elemente (și nu numai prin cei cu k de 1), timpul total de lucru este redus, deoarece pentru vectorii care nu au exact k de 1 funcția nu se va evalua.

La evaluare, pentru ca un anumit produs de coeficient 1 să fie 1 este necesar ca toate componentele sale să fie 1. Numărul de ordine al produsului trebuie să fie deci conținut în reprezentarea binară a numărului de ordine al vectorului (ținând cont de generarea prin numărare și de toți cei 2^n vectori). Excepție face cazul în care a_0 este 1, deci valoarea funcției este 1 pentru orice vector.

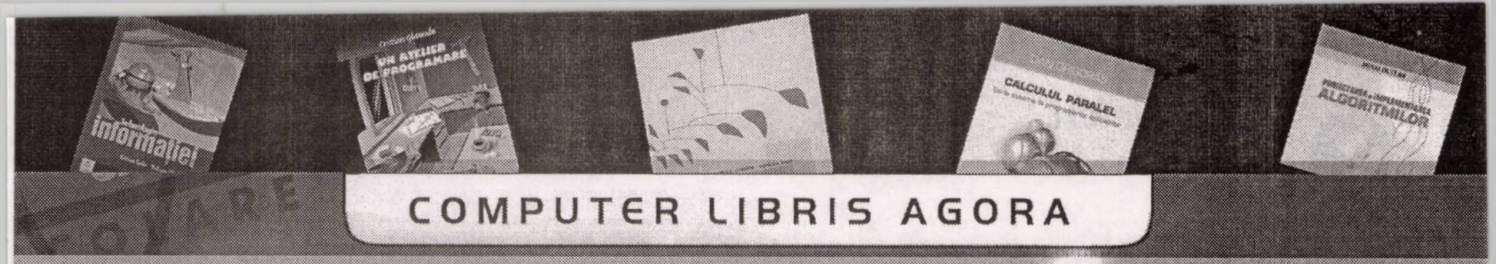
În momentul în care unul dintre produsele cu coeficienți 1 este 1, valoarea funcției este 1, deci se întrerupe evaluarea. Aceasta face ca evaluarea să fie, în general, destul de rapidă, deși teoretic ordinul de complexitate al acestei operații este $O(2^n)$.

Analiza complexității

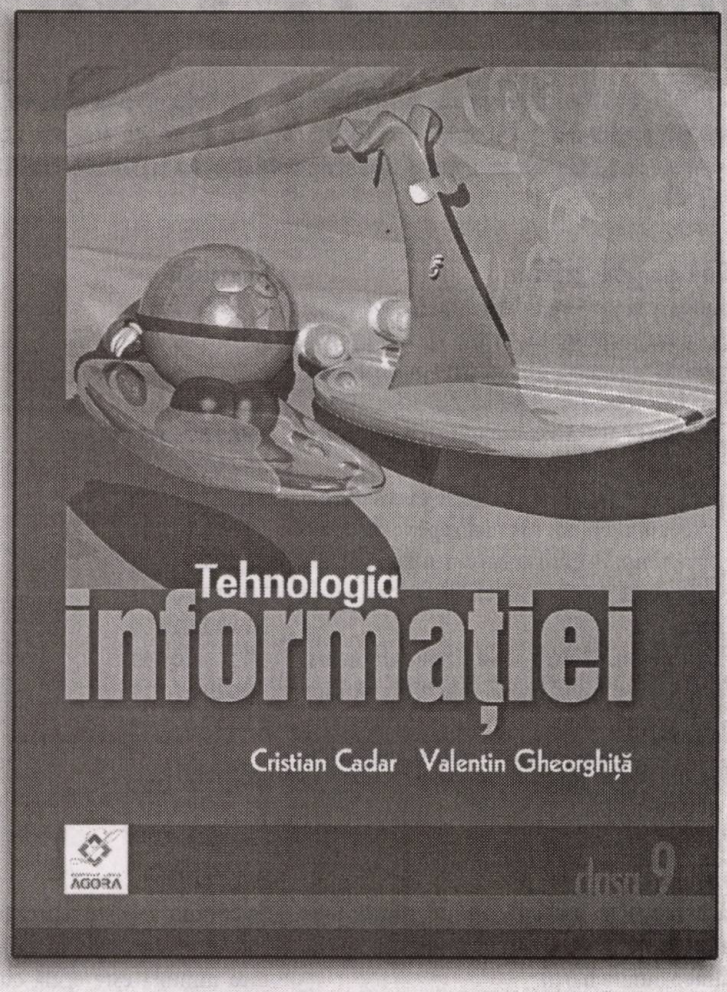
Operația de citire a datelor de intrare are ordinul de complexitate $O(2^n)$, iar scrierea datelor are ordinul de complexitate $O(1)$.

Pentru fiecare vector, evaluarea are ordinul de complexitate, teoretic, $O(2^n)$. În total sunt $C(n, k)$ vectori.

Ordinul total de complexitate al algoritmului este, teoretic, $O(2^n \cdot C(n, k))$. Datorită întreruperii evaluării, un program care implementează această rezolvare va rula destul de rapid.



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 0264 - 592422

COMPUTER LIBRIS AGORA

Rundele #01 - #05

Cea de-a V-a ediție a concursului internațional de programare Bursele Agora a început pe data de 3 decembrie 2003. În continuare vă prezentăm enunțurile problemelor propuse spre rezolvare la primele cinci runde ale acestei ediții.

P080313: Cei șapte pitici

Probabil că toți cunoașteți povestea "*Albă ca Zăpada și cei șapte pitici*". Ceea ce nu știți este că, într-o zi, când piticii lucrau în mină a avut loc un cutremur și toate puțurile care coborau în mină au fost distruse. *Albă ca Zăpada*, văzând că piticii nu au ajuns seara acasă s-a hotărât să meargă să-i caute.

Ajungând la mină, și-a dat seama că nu se putea coborî și nu știa cum să-i scoată pe pitici cât mai repede de acolo deoarece trecuse ceva timp. A găsit o hartă a minei pe care erau marcate zonele unde lucrau piticii și o mașină de săpat puțuri verticale.

Presupunând că niciuna dintre aceste zone nu a fost distrusă de cutremur, *Albă ca Zăpada* a hotărât să sape puțuri pentru a-i elibera pe pitici.

Ajutați-o pe *Albă ca Zăpada* să determine o modalitate de a săpa puțuri de lungime minimă pentru a-i scoate pe pitici din mină.

Date de intrare

Fișierul de intrare `DWARFS.IN` conține pe prima linie două numere naturale M și N separate printr-un spațiu care reprezintă dimensiunile hărții.

Pe fiecare dintre următoarele M linii se află un șir de N numere care pot fi `0` sau `1` și care nu sunt separate prin spații.

Cifra `1` indică faptul că pe hartă, acea celulă este acoperită de pământ, iar cifra `0` indică faptul că acea celulă nu este acoperită.

O zonă în care se pot afla pitici este formată din celule adiacente pe verticală sau orizontală care au valoarea `0` pe hartă.

Pe hartă, în locul puțurilor a fost trecută valoarea `1`, deoarece acestea au fost distruse.

Date de ieșire

Fișierul de ieșire `DWARFS.OUT` trebuie să conțină un singur număr natural L care reprezintă suma lungimilor tuturor puțurilor săpate. Această sumă trebuie să fie cât mai mică posibil.

Restricții

- $5 \le N \le 20$;
- $5 \le M \le 20$.

Exemplu

```
DWARFS.IN
8 10
1111111111
1101111111
1100101111
1001100101
1111111101
1011110001
1001111111
1111111111

DWARFS.OUT
5
```

Explicație

În continuare este reprezentată harta care conține caracterul `'*'` în locurile în care s-au săpat puțuri. Deci, lungimea minimă este egală cu numărul de celule marcate cu `'*'`.

```
11*11*1111
11011*1111
1100101111
1001100101
1*1111*101
1011110001
1001111111
1111111111
```

Timp maxim de execuție/test: 1 secundă

P080314: Asteroizi

Nava stelară *USS Enterprise* a fost singura care a supraviețuit confruntării cu borgii din sectorul 001 (*Terra*). Din fericire, uriașa navă a borgilor a fost distrusă, așadar cel

mai important sector al spațiului controlat de *Federația Planetelor Unite* nu a fost ocupat de borgi.

Totuși, *Enterprise* este grav avariată și trebuie să ajungă la șantierul naval din apropierea *Lunii*. Bătălia finală a avut loc în apropierea planetei *Uranus*, deci *Cpt. Jean Luc Picard* și restul echipajului care a supraviețuit luptei, trebuie să conducă nava prin centura de asteroizi aflată între *Marte* și *Jupiter*.

Din nefericire, manevrabilitatea navei este redusă, iar o schimbare de direcție este foarte dificil de realizat. Din acest motiv *Cpt. Jean Luc Picard* îi cere *Lt. Cmdr. Data* să stabilească un curs de zbor prin această centură de asteroizi, astfel încât să fie necesare cât mai puține schimbări de direcție.

Centura de asteroizi poate fi privită ca fiind o matrice bidimensională ale cărei elemente pot avea fie valoarea `0` (corespunzătoare unei zone libere), fie valoarea `1` (corespunzătoare unei zone în care se află un asteroid).

Navigatorul, *Slt. Wesley Crusher*, poate intra în centura de asteroizi prin orice punct dorește (va ajunge pe prima linie a matricei având direcția pe care o dorește; așadar, nu va fi efectuat nici un viraj pentru intrarea în centură). El trebuie să ajungă pe ultima linie a matricei, iar pentru a ieși din centură nu este necesară efectuarea unui viraj suplimentar.

USS Enterprise se poate deplasa prin centura de asteroizi pe verticală, orizontală sau diagonală. Oricare tip de viraj este la fel de dificil de efectuat; pot fi realizate viraje de 45°, 90° sau 135°, înspre stânga sau înspre dreapta. Nava stelară nu va putea trece prin zonele în care se află asteroizi.

După ce *Lt. Cmdr. Data* stabilește cursul de zbor, *Lt. Cmdr. William Riker* îl întreabă care este numărul virajelor care vor trebui efectuate.

La această întrebare va trebui să răspundeți și voi, cunoscând configurația centurii de asteroizi.

Date de intrare

Fișierul de intrare **`ASTEROID.IN`** conține pe prima linie două numere naturale M și N separate printr-un spațiu care reprezintă dimensiunile centurii de asteroizi.

Pe fiecare dintre următoarele M linii se află un șir de N numere care pot fi `0` sau `1` și care nu sunt separate prin spații.

Date de ieșire

Fișierul de ieșire **`ASTEROID.OUT`** trebuie să conțină un singur număr natural V care reprezintă numărul minim al virajelor care trebuie efectuate.

Restricții și precizări

- $5 \leq N \leq 200$;
- $5 \leq M \leq 200$;
- elementele de pe prima coloană și de pe ultima coloană ale matricei care reprezintă centura de asteroizi vor avea întotdeauna valoarea `1`.

Exemplu

`ASTEROID.IN`

```
8 10
1101111011
1100001101
1100101111
1001100101
1110111001
1010110001
1001111001
1101110011
```

`ASTEROID.OUT`

```
3
```

Explicație

În continuare este reprezentat traseul navei; pozițiile prin care trece nava sunt marcate cu `'*'`. Caracterele îngroșate indică poziții în care se efectuează viraje.

```
11*1111011
110**01101
11001*1111
100110*101
1110111*01
1010110*01
1001111*01
1101110*11
```

Există mai multe alte trasee pentru care sunt necesare tot trei viraje. Trei dintre ele sunt următoarele:

```
11*1111011    11*1111011    11*1111011
11*0*01101    11***01101    11***01101
11**1*1111    11001*1111    11001*1111
10*110*101    100110*101    100110*101
1110111*01    1110111*01    1110111*01
1010110*01    1010110*01    10101100*1
1001111*01    1001111*01    1001111*01
1101110*11    1101110*11    110111*011
```

Se observă că un curs de zbor cu număr minim de viraje nu este neapărat cel mai scurt curs. De exemplu, cursul prezentat în continuare este mai scurt decât oricare dintre cele prezentate anterior, dar necesită patru viraje.

```
11*1111011
11*0001101
11*0101111
10*1100101
111*111001
101*110001
10*1111001
11*1110011
```

Timp maxim de execuție/test: 1 secundă

P080315: Recurență

Șirul lui *Fibonacci* este una dintre cele mai celebre recurențe pe care o întâlnesc informaticienii. El este definit prin:

- $F_0 = 0$;
- $F_1 = 1$;
- $F_i = F_{i-1} + F_{i-2}$, pentru $i \geq 2$.

Așadar, primele elemente ale acestui șir sunt:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Putem defini șirul $R_i = F_{i+1} / F_i$. Primele elemente ale acestui șir vor fi:

```
1/0 1/1 2/1 3/2 5/3 8/5 13/8 21/13 ...
```

sau:

```
∞ 1 2 1.5 1.(6) 1.6 1.625 1.(615384) ...
```

Există o mulțime de șiruri care pot fi definite printr-o metodă asemănătoare. Relația de recurență este aceeași, dar valorile primilor doi termeni sunt arbitrare. Forma generală a acestor șiruri este:

- $a_0 = p$;
- $a_1 = q$;
- $a_i = a_{i-1} + a_{i-2}$, pentru $i \geq 2$.

Evident și pentru astfel de șiruri vom putea defini șiruri de forma $r_i = a_{i+1} / a_i$.

De exemplu, pentru $p = 4$ și $q = 5$ primele elemente ale șirului vor fi:

```
4 5 9 14 23 37 60 97 157 254 411 665 1076
```

În aceste condiții, elementele șirului $r_i = a_{i+1} / a_i$ vor fi:

```
5/4 9/5 14/9 23/14 37/23 60/37 97/60 ...
```

sau:

```
1.25 1.8 1.(5) 1.6(428571)
1.(6086956521739130434782) 1.(621)
1.61(6) ...
```

Cunoscându-se valorile elementelor p, q și n, va trebui să determinați valoarea r_n.

Date de intrare

Fișierul de intrare **REC.IN** conține trei linii. Pe prima linie se află valoarea p, pe cea de-a doua linie se află valoarea q, iar pe a treia se află valoarea n.

Date de ieșire

Fișierul de ieșire **REC.OUT** trebuie să conțină o singură linie pe care se va afla valoarea r_n, scrisă folosind opt zecimale exacte.

Restricții și precizări

- $1 \leq p, q \leq 1000$;
- p și q sunt numere reale care conțin cel mult două zecimale;
- $0 \leq n < 10^{100}$ (valoarea n este un număr format din cel mult 100 de cifre).

Exemple

REC.IN
```
1
1
2
```

REC.OUT
```
1.50000000
```

REC.IN
```
4
5
6
```

REC.OUT
```
1.616666666
```

REC.IN
```
24
57
0
```

REC.OUT
```
2.37500000
```

Explicații

Șirul din primul exemplu este, practic, șirul lui *Fibonacci* din care a fost eliminat primul element, iar toate celelalte elemente au fost deplasate cu o poziție. Așadar, elementul căutat, r_2, este, de fapt, elementul R_3, a cărui valoare este `1.5`.

Șirul corespunzător celui de-al doilea exemplu este cel folosit pentru exemplificările din enunț. Valoarea elementului căutat este, așa cum se poate vedea în descrierea problemei, `1.61(6)`.

Pentru cel de-al treilea exemplu, trebuie să determinăm valoarea r_0. Avem:

$$r_0 = a_1 / a_0 = q / p = 57 / 24 = 2.375.$$

Timp maxim de execuție/test: 1 secundă

P080316: Parada piticilor

Pe data de 13 *Decembrie* se serbează *Ziua Națională a Țării Piticilor*. Cu această ocazie va avea loc *Parada Militară a Armatei Piticilor*.

Deoarece se cunoaște preferința pentru simetrie a *Piticului Șef*, conducătorul paradei militare, *Generalul Pitic*, știe că soldații pitici vor trebui să defileze în șir indian și culorile scufiilor piticilor trebuie să urmeze în aceeași succesiune indiferent de direcția din care sunt priviți piticii care defilează.

Cu alte cuvinte, scufia primului pitic trebuie să aibă aceeași culoare ca scufia ultimului pitic, scufia celui de-al doilea pitic trebuie să aibă aceeași culoare cu cea a penultimului pitic și așa mai departe.

După ce și-a aranjat soldații astfel încât această cerință să fie respectată, *Generalul Pitic*, a plecat să stabilească ultimele detalii pentru ca parada să se desfășoare în cele mai bune condiții.

Din nefericire, în acest timp, soldații au fost vizitați de *Miss Pitic* și, în momentul în care au văzut-o, aceștia au rupt imediat rândurile (de fapt, rândul).

Când au auzit că *Generalul Pitic* se întoarce, piticii au format repede un nou rând dar, evident, acesta nu mai era simetric.

Generalul Pitic s-a întors și a observat dezastrul produs; s-a alarmat și a vrut să anuleze parada datorită faptului că și-a dat seama că în timpul rămas nu va reuși să rearanjeze soldații pitici.

Din fericire, consilierului său, *Colonelul Pitic*, i-a venit o idee care ar putea salva situația. Soluția sa este ca anumiți pitici să iasă din rând și rândul rămas să fie simetric.

Generalul Pitic a agreat această idee (de fapt, nu prea avea de ales) și i-a cerut unui pitic de încredere (*Caporalul Pitic*), să aleagă piticii care trebuie să iasă din rând pentru ca șirul simetric obținut să fie cât mai lung posibil (dorea ca parada să dureze cât mai mult, chiar și în aceste condiții).

Așadar, cunoscând succesiunea culorilor scufiilor din rândul pe care l-au format soldații după vizita lui *Miss Pitic*, va trebui să determinați numărul minim de pitici care trebuie să părăsească șirul pentru ca acesta să rămână simetric.

Date de intrare

Prima linie a fișierul de intrare **PARADE.IN** conține numărul N al piticilor care ar fi trebuit să participe la paradă. Pe fiecare dintre următoarele N linii se află câte un șir de caractere care reprezintă culoarea scufiei unui pitic din șir. Ordinea în care sunt descrise culorile scufiilor este dată de ordinea în care se află piticii după vizita lui *Miss Pitic*.

Date de ieșire

Fișierul de ieșire **PARADE.OUT** trebuie să conțină o singură linie pe care se va afla numărul minim al piticilor care trebuie să părăsească șirul.

Restricții și precizări

- $2 \le N \le 200$;
- culorile piticilor sunt șiruri de caractere formate din cel mult zece litere ale alfabetului englezesc.

Exemplu

PARADE.IN	PARADE.OUT
5 red green blue blue red	1

Explicație

Doar piticul cu scufie verde (`green`) trebuie să părăsească șirul. Vor rămîne patru pitici; primul va avea scufie roșie (`red`), următorii doi scufie albastră (`blue`), iar ultimul sufie roșie (`red`). Așadar, șirul piticilor va fi simetric.

Timp maxim de execuție/test: 1 secundă

P080317: Pluto

Pe o tablă sunt desenate N cercuri. *Pluto* trebuie să deseneze un număr total de M săgeți, astfel încât fiecare săgeată să pornească de la un cerc și să ajungă la alt cerc. Pentru fiecare cerc se cunoaște numărul săgeților care trebuie să plece din cercul respectiv și numărul săgeților care trebuie să ajungă la cercul respectiv. Va trebui să verificați dacă *Pluto* poate desena săgețile și, dacă acest lucru este posibil, să descrieți modul în care acestea vor fi desenate.

Date de intrare

Prima linie a fișierului de intrare **PLUTO.IN** conține numărul N al cercurilor și numărul M al săgeților. Cea de-a doua linie conține N numere întregi, separate prin câte un spațiu, care reprezintă numărul săgeților care trebuie să pornească de la fiecare cerc. Cea de-a treia linie conține N numere întregi, separate prin câte un spațiu, care reprezintă numărul săgeților care trebuie să ajungă la fiecare cerc. Primul număr de pe fiecare dintre aceste linii corespunde primului cerc (identificat prin 1), al doilea număr corespunde celui de-al doilea cerc (identificat prin 2) etc. Suma numerelor de pe fiecare dintre aceste două linii este întotdeauna M.

Date de ieșire

Prima linie a fișierului de ieșire **PLUTO.OUT** va conține mesajul `YES` în cazul în care există cel puțin o posibilitate de desenare a săgeților și mesajul `NO` în caz contrar. În cazul în care săgețile pot fi desenate, fișierul va mai conține M linii pe care se vor afla câte două numere întregi x și y cu semnificația: *va fi desenată o săgeată care pornește de la cercul identificat prin x și ajunge la cercul identificat prin y.*

Restricții și precizări

- $1 \le N \le 50$;
- $1 \le M \le 500$;
- cercurile sunt identificate prin numere naturale cuprinse între 1 și N;
- poate fi desenată cel mult o săgeată care pornește de la un cerc x și ajunge la un cerc y;
- dacă s-a desenat o săgeată care pornește de la un cerc x și ajunge la un cerc y, atunci poate sau nu fi desenată și o săgeată care pornește de la cercul y și ajunge la cercul x;
- dacă există mai multe soluții atunci poate fi aleasă oricare dintre ele.

Exemplu

PLUTO.IN	PLUTO.OUT
4 5 1 2 2 0 1 1 1 2	DA 1 2 2 3 3 1 2 4 3 4

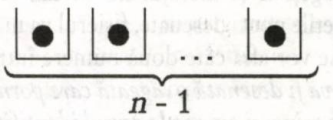
Timp maxim de execuție/test: 1 secundă

Principiul CUTIEI LUI DIRICHELET

Mihaela Oltean

Chiar dacă principiul cutiei lui Dirichelet se bazează pe una dintre cele mai simple observații matematice, rezolvarea problemelor folosind această metodă nu este o sarcină prea ușoară. În acest articol vom prezenta câteva probleme dificile care pot fi rezolvate folosind principiul cutiei lui Dirichlet.

Principiul *cutiei lui Dirichlet* se bazează pe una dintre cele mai simple observații matematice: *dacă avem n obiecte dispuse în n - 1 cutii, atunci există cel puțin o cutie care conține două obiecte*. Dar, pe cât de simplu este acest principiu, pe atât de complexe sunt implicațiile lui.



Există destule probleme, unele dintre ele propuse spre rezolvare la diferite concursuri de programare naționale sau internaționale, a căror soluție se poate obține mult mai ușor dacă se folosește principiul *cutiei lui Dirichelet*.

În continuare vom prezenta câteva probleme care se rezolvă folosind această observație matematică.

Problema 1

Se consideră cinci puncte în interiorul unui triunghi echilateral de latură o unitate. Se cere să se arate că există două puncte situate la o distanță de cel mult 0.5 unități.

Soluție

La început vom trasa cele trei linii mijlocii ale triunghiului. În acest fel triunghiul dat este împărțit în patru triunghiuri echilaterale identice. Laturile acestor triunghiuri vor avea lungimea 1/2 unități, deci două puncte aflate în același triunghi nu se pot afla la o distanță mai mare de 1/2 unități unul față de celălalt.

Avem, deci, patru triunghiuri și cinci puncte. Conform principiului *cutiei lui Dirichelet*, două puncte se vor găsi în același triunghi, deci se vor afla la o distanță de cel mult 1/2 unități unul față de celălalt.

O generalizare a acestei probleme este următoarea:

Fie un triunghi echilateral de latură n unități. Să se arate că oricum am alege $n^2 + 1$ puncte în interiorul triunghiului vor exista cel puțin două puncte la o distanță de cel mult o unitate.

Problema 2

Vom considera un pătrat și nouă linii, fiecare dintre ele tăind pătratul dat în două patrulatere de arii proporționale cu 2/3. Să se arate că există trei linii care trec prin același punct.

Soluție

Vom eticheta vârfurile pătratului cu *A*, *B*, *C*, *D* în sensul acelor de ceasornic.

Observăm, în primul rând, că nici una dintre linii nu poate tăia două laturi adiacente ale pătratului pentru că în felul acesta am obține un triunghi și un pentagon și în nici un caz un patrulater. Să presupunem că una dintre linii intersectează laturile *BC* și *AD* în punctele *M*, respectiv *N*. Patrulaterele *ABMN* și *CDNM* sunt, evident, trapeze având aceeași înălțime. Ariile lor sunt în același raport în care se află și liniile lor mijlocii. Deci, *MN* împarte linia mijlocie a pătratului în două segmente având raportul 2/3. Acest lucru este adevărat pentru oricare dintre cele nouă linii. Dar există doar patru puncte care împart liniile mijlocii ale pătratului în raportul 2/3. Conform principului cutiei, cel puțin trei linii trec prin același punct.

Problema 3

Considerăm 25 de puncte în plan cu proprietatea că în orice triplet există o pereche de puncte aflate la o distanță mai mică de o unitate. Să se arate că există un cerc de rază o unitate care conține 13 puncte dintre cele 25.

Soluție

Pentru a rezolva această problemă, vom alege la întâmplare un punct (pe care îl notăm cu A) dintre cele 25 date și vom considera cercul $B(A, 1)$ având centrul în A și raza o unitate.

Dacă toate punctele (sau cel puțin 13) sunt în B, atunci problema este rezolvată.

În caz contrar, considerăm un alt punct D care nu se află în B (distanța între A și D este mai mare de o unitate). Pentru orice alt punct C, fie distanța dintre A și C este mai mică de o unitate, fie distanța dintre D și C este mai mică de o unitate. Cu alte cuvinte, fiecare dintre punctele rămase aparțin fie cercului cu centrul în A și rază o unitate, fie cercului cu centrul în D și rază o unitate. Conform principiului cutiei, în interiorul unuia dintre cercuri se vor afla cel puțin 13 puncte.

Problema 4

Fie cinci puncte în plan, având coordonate întregi. Să se arate că unul dintre mijloacele segmentelor care unesc cele cinci puncte are, de asemenea, coordonatele întregi.

Soluție

Fiecare punct din plan este caracterizat de perechea (x, y), unde x este coordonata sa pe axa Ox, iar y este coordonata sa pe axa Oy.

Conform principiului *cutiei lui Dirichelet*, dintre cele cinci coordonate pe axa Ox (ale punctelor date) trei dintre ele au aceeași paritate.

Apoi, dintre cele trei puncte care au coordonatele pe Ox de aceeași paritate, vor exista două (tot conform principiului cutiei) pentru care coordonatele pe axa Oy au aceeași paritate. Mijlocul segmentului care unește aceste două puncte are coordonate numere întregi, deoarece media aritmetică a două numere de aceeași paritate este întotdeauna un număr întreg.

O altă rezolvare pentru această problemă poate fi următoarea: există patru variante pentru alegerea coordonatelor celor cinci puncte deoarece un punct poate să aibă coordonatele *pare* sau *impare*. Deci, conform principiului cutiei, există două puncte care au coordonatele alese în același fel (având aceeași paritate și pentru x și pentru y). În concluzie, mijlocul segmentului care leagă cele două puncte are coordonatele întregi.

Problema 5

Fie $P(x)$ un polinom cu coeficienți întregi. Dacă $P(a) = P(b) = P(c)$ pentru trei valori diferite a, b, c, să se arate că nu există d astfel încât $P(d) = 3$.

Soluție

Considerăm polinomul: $P(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$.

Propoziție:

Pentru oricare două numere întregi p și q, $P(p) - P(q)$ este divizibil prin $p - q$: $(p - q) \mid P(p) - P(q))$.

Demonstrație:

Într-adevăr, $P(p) - P(q) = c_n(p^n - q^n) + c_{n-1}(p^{n-1} - q^{n-1}) + \ldots + c_1(p - q)$ și deoarece $(p - q) \mid (p^k - q^k)$ pentru orice valoare $k > 0$, propoziția este demonstrată.

Cu această propoziție demonstrată, în continuare presupunem că: $P(a) = P(b) = P(c) = 2$ și $P(d) = 3$, unde a, b, c și d sunt distincte.

Din propoziție avem:

$(d - a) \mid (P(d) - P(a)) = 3 - 2 = 1$,
$(d - b) \mid (P(d) - P(b)) = 3 - 2 = 1$ și
$(d - c) \mid (P(d) - P(c)) = 3 - 2 = 1$,

deci $d - a$, $d - b$, $d - c$ sunt distincte și divid pe 1. Dar acest lucru nu este posibil (conform principiului cutiei) deoarece 1 are doar doi divizori distincți (pe 1 și pe -1), deci, presupunerea făcută este falsă. Astfel am demonstrat că nu există d astfel încât $P(d) = 3$.

Problema 6

Se consideră două numere naturale n și m. Se cere să se determine dacă există două numere naturale p și q distincte, astfel încât $m^p - m^q$ să fie divizibil cu n.

Soluție

Considerăm numerele $m^0, m^1, m^2, \ldots, m^n$. În total sunt $n + 1$ numere. Resturile împărțirii acestor numere la n sunt din mulțimea $\{0, 1, \ldots, n - 1\}$. Există în total n valori distincte pentru resturi, care sunt generate de $n + 1$ numere, deci (conform principiului *cutiei lui Dirichlet*) vor exista două numere distincte m^p și m^q care dau același rest prin împărțire la n, deci $m^p - m^q$ este divizibil cu n.

Algoritmul în pseudocod care rezolvă această problemă este:

```
Algoritm resturi
 citește n,m
 { vom folosi un tablou care are dimensiunea n, în care
   vom reține resturile și care conține resturile împărțirii
   valorilor m^i, i = 0, 1, ..., n, la n }
  resturi_0 ← 1;                    // rest[m^0/n] = 0
 pentru i=1,n execută:
  { m^(i-1) = n · c_(i-1) + resturi_(i-1)
   m^i = m · (n · c_(i-1) + resturi_(i-1)) = n · c_i + resturi_i
   ⇒ resturi_i = rest[(m · resturi_(i-1))/n] }
     resturi_i ← rest[(m · resturi_(i-1))/n]
 sfârșit pentru
  // căutăm două resturi egale
 pentru p=0,n-1 execută:
    pentru q=1,n execută:
       dacă resturi_p = resturi_q atunci
          scrie "Numerele căutate sunt ",p," și ",q
          *ieșire
       sfârșit dacă
    sfârșit pentru
 sfârșit pentru
sfârșit algoritm
```

Problema 7

Se consideră un număr natural p care nu este divizibil cu 10. Să se determine dacă există o putere a lui p care se termină cu 0...01 (k valori egale cu zero).

Soluție

Fie p^m și p^n ($m < n$) două puteri ale lui p care dau același rest prin împărțire la 10^{k+1} (vezi problema 6). Atunci $p^n - p^m$ este divizibil cu 10^{k+1}.

Știm că $p^n - p^m = p^m(p^{n-m} - 1)$ divizibil cu 10^{k+1}.

Dar p^m nu se divide cu 10. Înseamnă că $p^{n-m} - 1$ se divide cu 10^{k+1}. Deci $p^{n-m} - 1$ se termină cu $k + 1$ cifre de zero, iar p^{n-m} se termină cu k cifre egale cu 0 și un 1.

Problema 8

Se consideră un cerc cu raza de o unitate și șase puncte în interiorul său. Să se arate că există două puncte la o distanță de cel mult o unitate.

Soluție

Împărțim cercul în șase sectoare egale astfel încât unul dintre punctele date (să-l notăm pe acesta cu A) să se afle pe una dintre cele șase raze care unesc centrul cu circumferința sa.

Distanța între oricare două puncte din același sector este cel mult o unitate. Dacă într-unul dintre cele două sectoare în care este inclus punctul A se mai află încă un punct, atunci problema este rezolvată. În caz contrar, mai rămân patru sectoare în care sunt dispuse cinci puncte și, conform principiului cutiei, va exista sigur un sector în care vom avea două puncte.

Cele două puncte din același sector se află la o distanță mai mică sau egală cu o unitate unul față de celălalt.

Problema 9

Se consideră un număr $n > 3$. Fie m cel mai mare număr întreg mai mic decât $(n + 2) / 2$. Să se arate că, alegând la întâmplare cel puțin m numere întregi din intervalul $[1, n]$, există trei numere (dintre cele alese) cu proprietatea că unul este egal cu suma celorlalte două.

Soluție

Vom considera că x este cel mai mic număr ales. Calculăm toate diferențele dintre x și celelalte numere alese. Vom avea m diferențe. În continuare vom arăta că una dintre aceste diferențe este egală cu unul din numerele selectate.

Evident, nu există două diferențe egale, deoarece în acest caz numerele selectate nu mai sunt distincte. Numerele neselectate sunt în număr de $n - m$. Avem, deci, m diferențe și $n - m$ valori pe care le-ar putea avea aceste diferențe. Dar $m > n - m$, deoarece $2 \cdot m > n$.

Problema 10

Considerăm un număr natural n. Să se arate că dacă se aleg mai mult de jumătate din întregii din intervalul $[1, 2 \cdot n]$, atunci vor exista doi întregi care se divid unul pe celălalt.

Soluție

Construim mulțimile $A_i = \{2^p \cdot (2 \cdot i - 1) \mid p = 0, 1, \ldots\}$, care conțin numerele impare împreună cu multiplii lor mai mici sau egali cu $2 \cdot n$. Mulțimea A_i conține al i-lea număr impar din intervalul $[1, 2 \cdot n]$. În intervalul $[1, 2 \cdot n]$ există n întregi impari, deci vom avea n mulțimi. Reuniunea acestor mulțimi este mulțimea $\{1, 2, \ldots, 2 \cdot n\}$.

Alegând $n + 1$ întregi, doi vor aparține aceleiași mulțimi. În aceste mulțimi există cel puțin două numere: un număr impar și dublul său care satisfac cerința din enunț.

Problema 11

Să se arate că dacă se aleg 55 de numere întregi distincte din intervalul $[1, 100]$, există o pereche de numere a căror diferență este 10, o altă pereche a cărei diferență este 12, dar nu există nici o pereche a cărei diferență să fie 11.

Soluție

Împărțim numerele între 1 și 20 în zece mulțimi: $\{1, 11\}$, $\{2, 12\}$, ..., $\{10, 20\}$. Continuăm împărțirea cu numerele din intervalele $[21, 40]$, $[41, 60]$, $[61, 80]$, $[81, 100]$. Avem în total 50 de mulțimi. Deci, dacă alegem 55 de numere, vor exista două din aceeași mulțime (care au diferența 10).

Pentru diferența 12 vom construi mulțimile: $\{1, 13\}$, $\{2, 14\}$, ..., $\{12, 24\}$. Vom face la fel și pentru celelalte numere. La sfârșit mai rămân mulțimile $\{97\}$, $\{98\}$, $\{99\}$, $\{100\}$. În final vom avea 52 de mulțimi. Dacă alegem 55 de numere, vor exista două din aceeași mulțime, adică vor avea diferența 12.

Dacă se încearcă dispunerea numerelor în mulțimi cu diferența 11, se va observa că se vor construi 55 de mulțimi, deci se vor putea alege 55 de numere a căror diferență să nu fie 11.

Problema 12

Demonstrați că printre oricare n numere întregi din intervalul $[1, 2 \cdot n - 2]$ există doi întregi care au suma impară.

Soluție

Construim mulțimile: $\{1, 2\}$, $\{3, 4\}$, ..., $\{2 \cdot n - 1, 2 \cdot n - 2\}$. Vom avea în total $n - 1$ mulțimi. Alegând n numere, vor exista două care aparțin aceleiași mulțimi. Suma acestor două numere este impară.

Problema 13

Se consideră 1000 de numere întregi distincte. Să se determine dacă există doi dintre ei care au diferența sau suma egală cu un multiplu de 1997.

Soluție

Construim mulțimile de resturi modulo 1997: $\{0\}$, $\{i, 1997 - i\}$, $1 \le i \le 998$. Toate numerele dau, prin împărțirea la 1997, un rest aflat în una din cele 999 de clase. Dacă două din 1000 de numere, prin împărțirea la 1997, vor da același rest, atunci ele vor diferi unul de celălalt printr-un multiplu de 1997. În caz contrar vor exista două numere care au

suma resturilor împărțirii la 1997 în aceeași mulțime, deci suma resturilor lor este 1997.

Problema 14

Într-o cameră se află n persoane. Fiecare dintre acestea cunoaște un număr de alte persoane din cameră. Se știe că relația de cunoștință este reciprocă (dacă A cunoaște pe B, atunci și B cunoaște pe A). Să se determine dacă există două persoane care cunosc același număr de persoane din cameră.

Soluție

Fiecare persoană din cameră cunoaște cel puțin 0 persoane și cel mult n - 1 persoane.

Dacă există o persoană care le cunoaște pe toate celelalte n - 1 persoane, atunci nu va exista o persoană care să nu cunoască nici o altă persoană din cameră.

Dacă există o persoană care nu cunoaște pe nimeni din cameră, atunci nu va exista o persoană care să cunoască pe toate celelalte n - 1 persoane din cameră.

Dacă vom construi mulțimea numerelor corespunzătoare numărului de persoane cunoscut de fiecare din cameră, atunci aceasta va conține cel mult n - 1 valori distincte (dacă îl conține pe 0, atunci nu îl conține pe n - 1, iar dacă îl conține pe n - 1 nu îl conține pe 0). Deoarece în cameră se află n persoane, înseamnă că cel puțin două dintre ele cunosc un același număr de persoane din cameră.

O altă variantă de enunț pentru această problemă ar fi următoarea: *Să se arate că într-un graf neorientat există două vârfuri care au același grad.*

Problema 15

La un campionat de fotbal participă n echipe care joacă între ele, astfel încât oricare două joacă o singură dată. Să se arate că în orice moment există două echipe care au jucat același număr de jocuri.

Soluție

La un moment dat fiecare echipă a jucat cel puțin 0 jocuri, dar nu mai mult de n - 1.

Dacă există o echipă care a jucat n - 1 jocuri, înseamnă că nu există o echipă care să nu fi jucat nici un joc. Dacă există o echipă care nu a jucat nici un joc, înseamnă că nici o echipă nu a jucat toate cele n - 1 jocurile. Dacă formăm mulțimea numerelor care constituie numărul de partide jucate de fiecare echipă, atunci aceasta va conține cel mult n - 1 numere (dacă îl conține pe 0, atunci nu îl conține pe n - 1, iar dacă îl conține pe n - 1 nu îl conține pe 0). Fiind n echipe, înseamnă că cel puțin două dintre ele au jucat același număr de jocuri.

Problema 16

Se consideră o mulțime de cifre C care conține și cifra 0. Să se determine un multiplu a lui n care este format din toate cifrele din C.

Soluție

Fie $C = \{c_1, c_2, ..., c_k\}$ mulțimea cifrelor date. Evident $2 \le k \le 10$. Presupunem, în continuare că, $c_k = 0$.

Construim numărul $A = c_1c_2...c_k$. Apoi construim șirul de numere:

$x_1 = A$,
$x_2 = AA$,
$x_3 = AAA$,
...
$x_n = A...A$ (A de n ori).

Calculăm resturile r_i ale împărțirii lui x_i la n. Aceste resturi sunt numere întregi din mulțimea $\{0, 1, ..., n - 1\}$. Dacă unul dintre resturi este 0, atunci numărul corespunzător este soluția problemei. În caz contrar vom avea n - 1 tipuri de resturi generate de n numere, deci, conform *principiului lui Dirichlet*, două resturi (r_i și r_j, $i < j$) sunt egale. Dacă scădem numerele corespunzătoare $x_j - x_i$, vom obține un număr, care prin împărțire la n, dă restul 0 și care reprezintă totodată și soluția problemei.

Problema 17

Fie A o mulțime cu n elemente. Să se arate că, dacă alegem mai mult decât jumătate dintre submulțimile mulțimii A, atunci două dintre acestea au proprietatea că una este inclusă în cealaltă.

Soluție

Se știe că o mulțime cu n elemente are 2^n submulțimi. Fie x un element din A și $B = A\setminus\{x\}$.

Pentru fiecare submulțime S a lui B vom forma perechea $\{S, S \cup \{x\}\}$. Aceste perechi formează o partiție a mulțimii A.

Dacă alegem mai mult de jumătate dintre submulțimile lui A, conform principiului cutiei, vor exista două care se află în aceeași pereche.

Două mulțimi din aceeași pereche au proprietatea că una este inclusă în cealaltă.

Bibliografie

1. Adrian Ghioca, Nicolae Teodorescu, *Culegere de probleme*, Editura SSMR, București, 1987.
2. Horia Georgescu, *Principiul cutiei lui Dirichlet, Gazeta de Informatică nr.* 9, 10, Editura Computer Press Agora, Cluj-Napoca, 1994
3. Nicolae Teodorescu, *Probleme din Gazeta de Matematică*, Editura Tehnică, București, 1984.
4. ***, *Colecția Gazeta Matematică*, Editura SSMR.
5. ***, *Colecția GInfo*, Editura Computer Libris Agora.
6. ***, *Probleme de matematică traduse din revista sovietică Kvant*, , Editura Didactică și Pedagogică, București, 1983.

Mihaela Oltean este cadru didactic la Colegiul "David Prodan" din Cugir
Poate fi contactată la adresa de e-mail olteanmihaelaelena@yahoo.com.

ALGORITMI - CALCULABILITATE

Horia Georgescu

În cadrul acestui articol vom aborda proprietățile de închidere ale funcțiilor și codificarea programelor cu ajutorul câtorva demonstrații și vom trata problema opririi programelor cu ajutorul tezei lui Church. Vom insista asupra calculabilității funcțiilor.

Proprietăți de închidere

Fie $g_1, ..., g_k$ funcții de aritate n, iar h o funcție de aritate k. Putem atunci defini funcția f de aritate n prin:

$$f(x_1, ..., x_n) = h(g_1(x_1, ..., x_n), ..., g_k(x_1, ..., x_n)).$$

Propoziția 1

Dacă $g_1, ..., g_k$, h sunt (parțial) calculabile, atunci și funcția f este (parțial) calculabilă.

Funcția f este calculată de următorul program:

```
z1 ← g1(x1, ..., xn)
...
zk ← gk(x1, ..., xn)
y ← h(z1, ..., zk)
```

Propoziția de mai sus arată că atât clasa funcțiilor parțial calculabile, cât și clasa funcțiilor calculabile, sunt închise la operația de compunere.

Fie n un număr natural și funcțiile $f: N^m \to N$, $g: N^{n+2} \to N$. Vom defini funcția $h: N^{n+1} \to N$ prin:

$$(1)\begin{cases} h(x_1, \text{K}, x_n, 0) = f(x_1, \text{K}, x_n) \\ h(x_1, \text{K}, x_n, t+1) = g(t, h(x_1, \text{K}, x_n, t), x_1, \text{K}, x_n), t \geq 0 \end{cases}$$

Propoziția 2

Dacă f și g sunt calculabile, atunci h este calculabilă. Într-adevăr, funcția h este calculată de programul:

```
    y ← f(x1, ..., xn)
[A] if xn+1 = 0 goto E
    y ← g(t, y, x1, ..., xn)
    t ← t + 1
    xn+1 ← xn+1
    goto A
```

Această propoziție demonstrează închiderea clasei funcțiilor calculabile la recursii de tipul (1).

Pentru cazurile $n = 0$ și $n = 1$, recursia (1) se scrie:

$$(1')\begin{cases} h(0) = k \in N \\ h(t+1) = g(t, h(t)) \end{cases}, \text{ respectiv}$$

$$(1'')\begin{cases} h(x,0) = f(x) \\ h(x,t+1) = g(t, h(x,t), x) \end{cases},$$

unde în (1') s-a ținut cont că funcția de aritate 0 se identifică cu o constantă.

Proprietățile de închidere luate în considerare mai sus ne permit să prezentăm și alte exemple de funcții calculabile; fiecare dintre ele permite introducerea unei macroinstrucțiuni, conform exemplului 7. Noile exemple ne vor fi utile în continuare și în plus ne vor mări încrederea în limbajul S.

Exemplul 9

Funcția $h(n) = n!$ este calculabilă.

Într-adevăr: $\begin{cases} h(0) = 1 \\ h(t+1) = h(t) \cdot (t+1) = g(t, h(t)), t > 0 \end{cases}$,

unde $g(x_1, x_2) = (x_1 + 1) \cdot x_2$.

Exemplul 10

Funcția $h: N \times N \to N$ dată de $h(x_1, x_2) = x_1^{x_2}$ este calculabilă.

Pentru demonstrație, plecăm de la relațiile: $x^0 = 1$; $x^{y+1} = x^y \cdot x$ (am luat deci prin convenție $0^0 = 1$), obținând:

$$\begin{cases} h(x_1,0) = 1 \\ h(x_1,t+1) = h(x_1,t) \cdot x_1 = g(t,h(x_1,t),x_1 \end{cases},$$

unde $g(x_1, x_2, x_3) = x_2 \cdot x_3$.

Exemplul 11

Funcția $f : N \times N \to N$ dată de:

$$f(x_1,x_2) = x_1 \dot{-} x_2 = \begin{cases} x_1 - x_2, x_1 \geq x_2 \\ 0, x_1 < x_2 \end{cases},$$

este calculabilă (demonstrația este propusă ca exercițiu).

Exemplul 12

Funcția $f : N \times N \to N$, $f(x_1, x_2) = |x_1 - x_2|$, este calculabilă, deoarece $|x_1 - x_2| = (x_1 \dot{-} x_2) + (x_2 \dot{-} x_1)$.

Exemplul 13

Funcția $\alpha : N \to N$, dată de:

$$\alpha(x) = \begin{cases} 1, x = 0 \\ 0, x \neq 0 \end{cases},$$

este calculabilă, așa cum s-a arătat în exemplul 8.

Exemplul 14

Predicatul $f : N \times N \to \{0, 1\}$ (predicatul "="), dat de:

$$f(x_1,x_2) = \begin{cases} 1, x_1 = x_2 \\ 0, \text{altfel} \end{cases},$$

este calculabil, deoarece $f(x_1, x_2) = \alpha(|x_1 - x_2|)$.

Exemplul 15

Predicatul $f : N \times N \to \{0, 1\}$ (predicatul "$\leq$"), dat de:

$$f(x_1,x_2) = \begin{cases} 1, x_1 \leq x_2 \\ 0, \text{altfel} \end{cases},$$

este calculabil, deoarece $f(x_1, x_2) = \alpha(x_1 \dot{-} x_2)$; într-adevăr, $x_1 \leq x_2 \Leftrightarrow x_1 \dot{-} x_2 = 0$.

Propoziția 3

Dacă P și Q sunt predicate calculabile, atunci $\neg P$, $P \vee Q$, $P \wedge Q$ sunt predicate calculabile.

Demonstrație

Este suficient să observăm că:
$\neg P = \alpha \circ P$; $P \wedge Q = P \cdot Q$; $P \vee Q = \neg((\neg P) \wedge (\neg Q))$.

Propoziția 4

Dacă g, h sunt funcții n-are (parțial) calculabile, iar P un predicat n-ar calculabil, atunci funcția n-ară:

$$f(x_1,x_2,\ldots,x_n) = \begin{cases} g(x_1,x_2,\ldots,x_n), P(x_1,x_2,\ldots,x_n) = 1 \\ h(x_1,x_2,\ldots,x_n), P(x_1,x_2,\ldots,x_n) = 0 \end{cases},$$

este (parțial) calculabilă (observăm analogia cu construcția **`if-then-else`**).

Demonstrație

Într-adevăr, $f = g \cdot P + h \cdot (\alpha \circ P)$.

Propoziția 5

Fie f o funcție calculabilă de aritate $n + 1$. Atunci funcțiile g și h definite de:

$$g(x_1,x_2,\ldots,x_n,x_{n+1}) = \sum_{t=0}^{x_{n+1}} f(x_1,x_2,\ldots,x_n,t)$$

$$h(x_1,x_2,\ldots,x_n,x_{n+1}) = \prod_{t=0}^{x_{n+1}} f(x_1,x_2,\ldots,x_n,t)$$

sunt calculabile.

Demonstrație

Este suficient să observăm că:

$$\begin{cases} g(x_1,x_2,\ldots,x_n,0) = f(x_1,x_2,\ldots,x_n,0) \\ g(x_1,x_2,\ldots,x_n,t+1) = g(x_1,x_2,\ldots,x_n,t) + \\ \qquad f(x_1,x_2,\ldots,x_n,t+1) \end{cases}$$

$$\begin{cases} h(x_1,x_2,\ldots,x_n,0) = f(x_1,x_2,\ldots,x_n,0) \\ h(x_1,x_2,\ldots,x_n,t+1) = h(x_1,x_2,\ldots,x_n,t) \cdot \\ \qquad f(x_1,x_2,\ldots,x_n,t+1) \end{cases}$$

și să aplicăm propoziția 2.

Propoziția 6

Fie P un predicat calculabil de aritate $n + 1$. Atunci sunt calculabile și următoarele predicate de aritate $n + 1$ cu definiție evidentă:

$$(\forall t)_{\leq n+1} P(x_1,x_2,\ldots,x_n,t); (\exists t)_{\leq n+1} P(x_1,x_2,\ldots,x_n,t);$$
$$(\forall t)_{<n+1} P(x_1,x_2,\ldots,x_n,t); (\exists t)_{<n+1} P(x_1,x_2,\ldots,x_n,t).$$

Demonstrație

Demonstrația se bazează pe următoarele echivalențe:

$$(\forall t)_{\leq n+1} P(x_1,x_2,\ldots,x_n,t) \Leftrightarrow \prod_{t=0}^{x_{n+1}} P(x_1,x_2,\ldots,x_n,t) = 1$$

$$(\exists t)_{\leq n+1} P(x_1,x_2,\ldots,x_n,t) \Leftrightarrow \sum_{t=0}^{x_{n+1}} P(x_1,x_2,\ldots,x_n,t) \neq 0$$

$$(\forall t)_{<n+1} P(x_1,\ldots,x_n,t) \Leftrightarrow (\forall t)_{\leq n+1}[t = x_{n+1} \vee P(x_1,\ldots,x_n,t)]$$

$$(\exists t)_{<n+1} P(x_1,\ldots,x_n,t) \Leftrightarrow (\exists t)_{\leq n+1}[t \neq x_{n+1} \wedge P(x_1,\ldots,x_n,t)]$$

Exemplul 16

Predicatul binar (notat prin "|") următor:

$$P(x_1, x_2) = 1 \Leftrightarrow x_1 \mid x_2,$$

este calculabil.

Demonstrație

Într-adevăr, $x_1 \mid x_2 \Leftrightarrow (\exists z)_{\leq x_2}(x_2 = x_1 \cdot z)$.

Exemplul 17

Predicatul unar *Prim*, definit prin:

$$Prim(x) = 1 \Leftrightarrow (x > 1) \wedge (\forall t)_{\leq x}[(t = 1) \vee \neg(t \mid x)],$$

este calculabil.

Propoziția 7

Fie P un predicat calculabil de aritate $n + 1$. Atunci următoarea funcție de aritate $n + 1$ este calculabilă:

$$\min_{t \le x_{n+1}} P(x_1, x_2, \ldots, x_n, t)$$

cu precizarea că dacă nu există un astfel de t, valoarea obținută va fi 0.

Demonstrație

Într-adevăr, funcția de mai sus este calculată de programul:

```
[A] if P(x1, ..., xn, t) goto B
    t ← t + 1
    if t ≤ xn+1 goto A
    y ← 0
    goto E
[B] y ← t
```

Rațiunea pentru care am ales prin lipsă valoarea 0 pentru minim va apărea în secțiunea următoare.

Exemplul 18

Funcția binară f ("partea întreagă din raport") dată de:

$$f(x_1, x_2) = \lfloor x_1/x_2 \rfloor$$

este calculabilă.

Demonstrație

Într-adevăr:

$$\lfloor x_1/x_2 \rfloor = \min_{t \le x_1} [(t+1) \cdot x_2 > x_1].$$

Să observăm că am acceptat că $\lfloor x/0 \rfloor = 0$.

Exemplul 19

Funcția binară R care calculează restul împărțirii a două numere naturale este calculabilă.

Demonstrație

Într-adevăr, $R(x_1, x_2) = x_1 \dot{-} (\lfloor x_1/x_2 \rfloor \cdot x_2)$. Să observăm că am acceptat că $R(x, 0) = x$.

Exemplul 20

Funcția unară definită prin:
$f(n)$ = al n-lea număr prim (dacă $n > 0$) sau 0 (dacă $n = 0$) este calculabilă.

Demonstrație

Vom nota $p_n = f(n)$. Deci $p_0 = 0$, $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, ...

Într-adevăr, pentru $n > 0$ avem:

$$p_{n+1} = \min_{t \le p_n! + 1} [Prim(t) \wedge (t > p_n)].$$

Această egalitate se bazează pe inegalitatea $p_{n+1} \le p_n! + 1$, care este adevărată deoarece fie $p_n! + 1$ este prim, fie are un divizor prim mai mic decât el care nu poate fi nici unul dintre $p_1, \ldots, p_n$.

Definim acum succesiv funcțiile h și k, evident calculabile, date de:

$$h(y, z) = \min_{t \le z} [Prim(t) \wedge (t > y)]$$
$$k(x) = h(x, x! + 1)$$

Rezultă:

$$\begin{cases} p_0 = 0 \\ p_{n+1} = k(p_n), \forall n \ge 0 \end{cases}$$

și, conform propoziției 2, funcția f este calculabilă.

Codificarea programelor

În această secțiune vom vedea cum pot fi codificate programele.

Reprezentarea perechilor și n-uplelor

Pentru orice două numere naturale $x, y \in N$, definim $\langle x, y \rangle = 2^x \cdot (2 \cdot y + 1) - 1$.

Funcția $f: N \times N \to N$ dată de $f(x, y) = \langle x, y \rangle$ este bijectivă.

Într-adevăr pentru orice z, există și sunt unice numerele x, y cu $\langle x, y \rangle = z$:

- x este cea mai mare putere a lui 2 care divide pe $z + 1$;
- y poate fi determinat de relația $2 \cdot y + 1 = (z + 1)/2^x$ (membrul drept al acestei relații fiind impar).

Notând prin l și r funcțiile prin care se obțin x și y din z (adică $\langle l(z), r(z) \rangle = z$), observăm că ele sunt calculabile deoarece:

$$\begin{cases} l(z) = \min_{x \le z} [\neg(2^{x+1} \mid (z+1))] \\ r(z) = \lfloor ((z+1)/2^{l(z)} - 1)/2 \rfloor. \end{cases}$$

Pentru reprezentarea n-uplului $(a_1, \ldots, a_n)$ definim *numărul Gödel* atașat astfel:

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} p_i^{a_i},$$

unde $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$ este șirul numerelor prime. De exemplu $[2, 0, 1] = 2^2 \cdot 5^1 = 20 = [2, 0, 1, 0, \ldots, 0]$.

Pentru orice $n \ge 1$, funcția n-ară f dată de $f(x_1, \ldots, x_n) = [x_1, \ldots, x_n]$ este evident calculabilă.

Din teorema fundamentală a aritmeticii deducem că:

- pentru orice x nenul, există $n, a_1, \ldots, a_n$ cu $[a_1, \ldots, a_n] = x$ (deci funcția f este surjectivă);
- din $[a_1, \ldots, a_n] = [b_1, \ldots, b_n]$ rezultă $a_i = b_i, \forall i = 1, \ldots, n$.

Mai observăm că $[a_1, \ldots, a_n] = [a_1, \ldots, a_n, 0]$. Prin urmare funcția care atașează oricărui program numărul său *Gödel* nu este injectivă.

De exemplu: $1 = [0] = [0, 0] = [0, 0, 0] = \ldots$; acest exemplu ne permite să definim numărul *Gödel* atașat secvenței vide (cu $n = 0$) ca fiind 1.

Pentru orice $i \ge 1$, considerăm funcția $(\ )_i: N \to N$ care pentru fiecare număr $x = [a_1, a_2, \ldots]$ calculează $(x)_i = a_i$. Aceste funcții sunt calculabile deoarece:

$$(x)_i = \min_{t \le x} [\neg(p_i^{t+1} \mid x)].$$

Să observăm că $(0)_i = (1)_i = 0, \forall i$.

Mai introducem funcția $Lt: N \to N$ care atașează fiecărui număr natural x numărul de ordine al celui mai mare număr prim care îl divide pe x. În exemplul de mai sus pentru $x = 20 = [2, 0, 1, 0, \ldots, 0]$ avem $Lt(x) = 3$.

Funcția Lt este calculabilă deoarece:

$$Lt(x) = \min_{i \le x} \left[(x)_i \neq 0 \wedge (\forall j)_{j \le x} \left((j \le i) \vee ((x)_j = 0) \right) \right].$$

(se caută cel mai mic i cu $x_i \neq 0$ și $(x)_j = 0$, $\forall j > i$). Să observăm că:

- $Lt(0) = Lt(1) = 0$;
- $Lt([a_1, \ldots, a_n]) = n \Leftrightarrow a_n \neq 0$ (unde $n \ge 1$);
- pentru orice $z \in N \setminus \{0, 1\}$, există și sunt unice $n, a_1, \ldots, a_n$ cu $a_n \neq 0$ și $[a_1, \ldots, a_n] = z$ (pentru $z = 1$ se obține $n = 0$, adică secvența vidă).

Numărul atașat unui program

Începem prin a așeza variabilele de intrare $(x_1, x_2, \ldots)$, variabila de ieșire y și variabilele de lucru $(z_1, z_2, \ldots)$ în următoarea ordine: $y, x_1, z_1, x_2, z_2, \ldots$

Variabilelor le atașăm pozițiile lor în acest șir prin funcția # astfel: $\#(y) = 1$; $\#(x_i) = 2 \cdot i$; $\#(z_i) = 2 \cdot i + 1$, $\forall i \ge 1$.

Fiecare instrucțiune poate avea atașată o etichetă. Vom numerota etichetele folosite în program cu $E_1, E_2, \ldots$. Extindem funcția # la etichete astfel: $\#(E_i) = i$.

În continuare observăm că o instrucțiune I în limbajul S este bine determinată de:

- etichetarea ei;
- tipul instrucțiunii;
- unica variabilă v care intervine în ea.

Corespunzător, pentru fiecare instrucțiune definim trei numere a, b și c astfel:

$$a = \begin{cases} 0, \text{dacă } I \text{ nu este etichetată} \\ \#(L), \text{dacă } I \text{ este etichetată cu } L \end{cases},$$

$$b = \begin{cases} 0, \text{dacă } I \text{ este de tipul } v \leftarrow v \\ 1, \text{dacă } I \text{ este de tipul } v \leftarrow v + 1 \\ 2, \text{dacă } I \text{ este de tipul } v \leftarrow v - 1 \\ \#(L) + 2, \text{dacă } I \text{ este de tipul } \textbf{if } v \neq 0 \textbf{ goto } L \end{cases},$$

$$c = \#(v) - 1,$$

unde v este unica variabilă care apare în instrucțiunea I.

Putem extinde acum funcția # la instrucțiuni astfel:
$\#(I) = \langle a, \langle b, c \rangle \rangle$ (numărul atașat instrucțiunii I).

Exemplul 21

Fie I următoarea instrucțiune:

$$[E_1] \quad x_1 \leftarrow x_1 + 1.$$

Atunci $a = 1, b = 1, c = 1$ și deci $\#(I) = \langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 21$.

Restricția funcției # la mulțimea instrucțiunilor, funcție având codomeniul N, este bijectivă, deoarece funcția $\langle , \rangle$ este bijectivă. Să mai observăm că unica instrucțiune I cu $\#(I) = 0$ este instrucțiunea neetichetată: $y \leftarrow y$.

Fie acum un program P constând, în ordine, din instrucțiunile $I_1, \ldots, I_n$. Extindem atunci funcția # la programe astfel: $\#(P) = [\#(I_1), \ldots, \#(I_n)] - 1$.

$\#(P)$ este numărul atașat programului P. Funcția # definită pe mulțimea programelor este calculabilă.

Ținând cont de observațiile făcute anterior asupra numărului *Gödel* atașat unei secvențe, rezultă că restricția funcției # la programe devine bijectivă, dacă ultima instrucțiune a programelor este diferită de instrucțiunea neetichetată $y \leftarrow y$; cum efectul acesteia este nul, vom impune (fără a scădea din generalitate) regula: *Nici un program nu se poate termina cu instrucțiunea neetichetată* $y \leftarrow y$.

În acest mod fiecare număr natural poate fi privit ca un (unic) program în limbajul S.

O consecință imediată este că mulțimea programelor în limbajul S este numărabilă; acest fapt se putea deduce și în alte moduri, dar cel de mai sus permite chiar "numerotarea programelor".

Exemplul 22

Căutăm programul P al cărui număr atașat este $\#(P) = 199$.

Observăm că: $199 + 1 = 200 = 2^3 \cdot 5^2 = [3, 0, 2]$, ceea ce arată că P este format din 3 instrucțiuni, ale căror numere atașate sunt în ordine 3, 0, 2:

$3 = \langle a, \langle b, c \rangle \rangle \Rightarrow a = 2$ și $\langle b, c \rangle = 0$, deci $b = c = 0$;

$2 = \langle a, \langle b, c \rangle \rangle \Rightarrow a = 0$ și $\langle b, c \rangle = 1$, deci $b = 1, c = 0$.

Rezultă că programul P căutat este următorul:

```
[E2] y ← y
     y ← y
     y ← y + 1
```

Să mai facem următoarele observații:

- programul vid are numărul atașat egal cu 1 - 1 = 0;
- corespondențele de mai sus sunt cele care au făcut necesare admiterea etichetării unor instrucțiuni diferite din același program cu aceeași etichetă.

Teza lui Church. Problema opririi

Teza lui *Church* (1936) se exprimă astfel: *Date fiind numerele naturale* $x_1, \ldots, x_n$, *numărul* y *poate fi "calculat" pe baza lor dacă și numai dacă există un program în limbajul* S *având la intrare valorile* $x_1, \ldots, x_n$ *și la ieșire valoarea* y.

Altfel spus, înțelegem prin algoritm care calculează valoarea y plecând de la valorile $x_1, \ldots, x_n$ un program în limbajul S care realizează acest lucru.

Evident, se pune problema dacă definiția cuvântului "algoritm" dată mai sus nu este prea restrictivă. Legat de aceasta, se impun următoarele precizări:

- noțiunea de algoritm nu poate fi definită decât pe baza unui limbaj de programare particular sau a unei "mașini matematice ideale"; de aceea teza lui *Church* nu poate fi

demonstrată ca o teoremă din matematică;
- toate încercările de a defini noțiunea de algoritm, încercări dintre care unele vor fi prezentate în continuare, au condus la definiții care s-au dovedit echivalente cu cea din enunțul tezei lui *Church*.

Aceste considerații ne determină să acceptăm definiția algoritmului așa cum apare ea în teza lui *Church*.

Suntem acum în măsură să prezentăm o primă problemă nedecidabilă, adică o problemă pentru care nu există un program în limbajul S care să o rezolve.

Este vorba de problema opririi (terminării) programelor.

Fie *HALT* predicatul binar definit astfel:

$HALT(x_1, x_2) = 1$, programul P cu $\#(P) = x_2$ se termină pentru valoarea de intrare x_1.

Teorema 1

Predicatul *HALT* nu este calculabil.

Demonstrație

Să presupunem prin absurd că predicatul *HALT* ar fi calculabil. Atunci putem considera următorul program (notat prin P) care constă din unica instrucțiune:

```
[A] if HALT(x,x) goto A.
```

Atunci funcția de un argument calculată de P este:

$$\psi_P^{(1)} = \begin{cases} \uparrow, \text{dacă} HALT(x,x) = 1 \\ 0, \text{dacă} HALT(x,x) = 0 \end{cases}.$$

Fie $y_0 = \#(P)$. Atunci: $HALT(x, y_0) = 1 \Leftrightarrow \psi_P^{(1)}(x)\downarrow$ (este nedefinită) $\Leftrightarrow HALT(x, x) = 0$ (programul cu numărul y_0 se termină pentru valoarea x).

Punând $x = y_0$, obținem $HALT(y_0, y_0) = 1 \Leftrightarrow HALT(y_0, y_0) = 0$. Se ajunge astfel la o contradicție.

Cum predicatul *HALT* nu este calculabil, conform tezei lui *Church* ajungem la următorul rezultat:

Corolar. Problema opririi programelor este nedecidabilă, în următorul sens: nu există un algoritm care, pentru orice program scris în limbajul S și orice valori de intrare, să decidă dacă programul se termină pentru acele date de intrare.

Va urma...

În următorul articol, care este de altfel și ultimul din această serie, vom prezenta alte probleme nedecidabile și vom trata problema decidabilității funcțiilor folosind diferite metode de abordare.

Bibliografie

1. Martin D. Davis, Elaine J. Weyuker, *Computability, Complexity and Languages*, Academic Press, 1983
2. Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994

Dl. prof. dr. Horia Georgescu este cadru didactic la Universitatea București, fondator și director științific al GInfo și poate fi contactat prin e-mail la adresa **g_horia@hotmail.com**.

WINAMP 5 (2+3)

Cea mai cunoscută și utilizată aplicație capabilă să ruleze muzică și clipuri video utilizând puține resurse sistem, Winamp, a ajuns la versiunea 5.

Acest lucru se datorează faptului că versiunea 3, chiar dacă s-a dorit să fie mai performantă decât versiunea 2, a fost considerată un eșec de cei de la NullSoft, mulți utilizatori ai acestui produs revenind la versiunea 2.

În primăvara acestui an a apărut prima versiune a produsului Winamp 5. Această versiune era *superprealpharelease* (m-am mirat foarte tare când am văzut numele versiunii) și nu avea nimic în plus față de produsul Winamp 2, din contră era plin de *bug*-uri.

De curând a apărut versiunea beta 10 care combină, într-un singur pachet, cele mai bune elemente prezente în cadrul celor două versiuni anterioare.

La facilitățile oferite de Winamp 2 și Winamp 3 s-au adăugat elemente noi, cum ar fi posibilitatea de a avea interfețe grafice cu altă formă decât cea cunoscută și ferestre translucide.

În momentul în care am pornit aplicația am rămas blocat. Nu am putut decât să-mi zic "Uaul".

Cu ajutorul acestui produs se pot viziona filme și videoclipuri și nu mai vorbesc de calitatea sunetului care este cunoscută chiar de la versiunea 1.

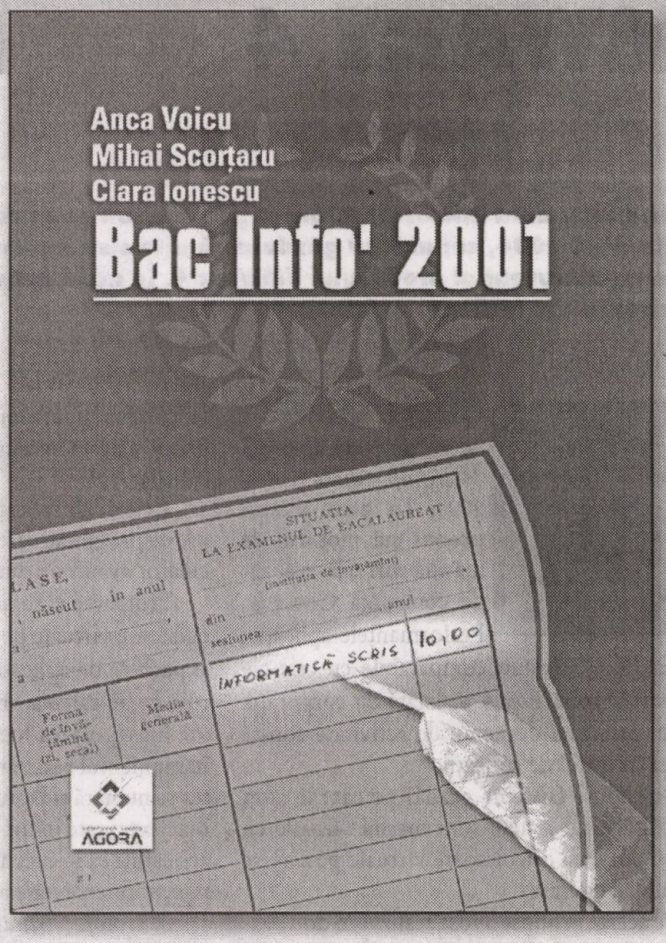
Fanii produselor Winamp au acum posibilitatea de a crea interfețe grafice care să aibă cele mai ciudate forme, Winamp 5 având inclus un pachet de *scripting* pentru a facilita acest lucru.

Cei de la NullSoft au anunțat adăugarea a două formate proprietare NSA (NullSoft Audio) și NSV (NullSoft Video).

Chiar dacă este încă la versiunea beta, acest produs merită să fie instalat. Mie mi-a plăcut.

Claudiu Soroiu

COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

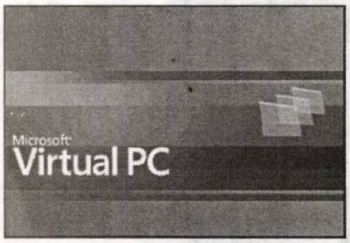
COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

VIRTUAL PC 2004 VS. VMWARE 4

Claudiu Soroiu

De curând compania Microsoft a lansat pe piață un nou produs și anume Microsoft Virtual PC 2004, capabil să găzduiască alte sisteme de operare. Acest produs este un concurent al produsului VMWare 4. În acest articol vom compara cele două produse din mai multe puncte de vedere.



Compania *Microsoft* a lansat pe piață aplicația *Microsoft Virtual PC* 2004. La baza acestui produs stă motorul de mașină virtuală realizat de compania *Conectix*. Performanțele aplicației *Microsoft Virtual PC* 2004 sunt comparabile cu cele ale aplicației *VMWare Workstation* 4 realizată de corporația *VMWare* al cărui principal domeniu de activitate constă chiar în realizarea de astfel de aplicații.

Aceste două produse (sisteme gazdă) pe care le vom compara în cadrul acestui articol sunt mașini virtuale cu ajutorul cărora se pot crea calculatoare virtuale pe care se pot instala sisteme de operare.

Fiecare dintre aceste produse a fost realizat în două variante: *workstation* (stație de lucru) și *server*. În acest articol vom compara doar versiunile *workstation*.

Ce este o mașină virtuală?

O *mașină virtuală* este o aplicație capabilă să interpreteze anumite instrucțiuni care rulează pe o mașină fizică (în zilele noastre calculator sau sistem de calcul). Instrucțiunile sunt interpretate cu ajutorul instrucțiunilor disponibile pe mașina fizică.

Pentru cunoscători, *un sistem de calcul* este un automat cu un anumit set de instrucțiuni, iar pentru toate automatele există un automat universal care să interpreteze instrucțiunile acestora.

Datorită faptului că un calculator face tot ce i se "dictează" putem spune că acesta este un automat universal care este capabil, prin intermediul programelor, să interpreteze instrucțiunile oricărei mașini. Deci se pot realiza programe care să execute instrucțiunile pe care le execută un calculator.

Cele două aplicații despre care vom discuta sunt exemple de programe care interpretează instrucțiunile unui calculator având o arhitectură 80x86.

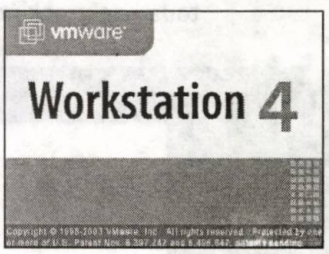
Programele de acest fel constituie, de fapt, niște calculatoare virtuale pe care se pot instala sisteme de operare și pachete de aplicații care rulează în același fel în care ar rula pe un calculator.

Există câteva lucruri care trebuie luate în considerare în momentul realizării unei astfel de aplicații. Printre acestea se numără și faptul că instrucțiunile unui calculator trebuie executate în timp real pentru ca sistemul de operare și aplicațiile să se execute optim. Deci, contează modul în care sunt interpretate aceste instrucțiuni.

Microsoft Virtual PC 2004 și *VMWare Workstation* 4 nu interpretează nici o instrucțiune 80x86. Instrucțiunile sunt transmise în mod direct procesorului căruia i se spune că modificările apărute în urma execuției nu se vor reflecta asupra sistemului de calcul, ci vor fi stocate într-o anumită zonă de memorie, iar datele cu care se operează sunt, de asemenea, stocate într-o zonă de memorie.

Generalități

Cele două aplicații pun la dispoziția mașinii virtuale pe care o creează toate componentele care pot fi găsite într-un calculator și, în afară de procesor, toate sunt virtuale.



Pentru ca să funcționeze, un calculator are nevoie de *placă de bază*, pro-

cesor, *memorie internă* și *memorie externă*, *placă video*, *monitor*, *tastatură*, *mouse* și, eventual, o *placă de rețea* pentru conectarea la alte calculatoare, o *placă de sunet* și/sau orice alt dispozitiv care este instalat pe sistemul de operare gazdă pentru care aplicațiile oferă suport virtual.

Ei bine, haideți să vedem modul în care aceste aplicații pun la dispoziția mașinilor virtuale componentele necesare pentru ca acestea să funcționeze.

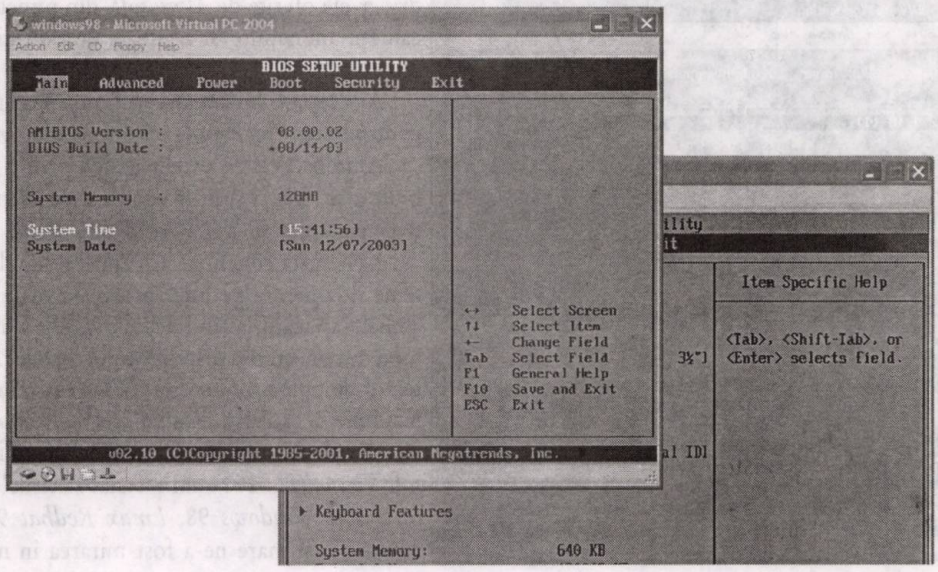
Am spus că procesorul interpretează direct aceste instrucțiuni.

Aplicațiile pun la dispoziția calculatoarelor virtuale un sistem virtual de intrare/ieșire astfel că:

- placa video pentru acestea este virtuală, iar monitorul este constituit dintr-o zonă din fereastra aplicației;
- în momentul în care zona de afișaj a conținutului din memoria video a calculatorului virtual devine activă, comenzile transmise de *mouse* și tastatură vor fi transmise acestuia; desigur că, nu toate combinațiile de taste vor fi active, deoarece unele combinații de taste sunt folosite de sistemul de operare gazdă pentru a executa anumite operații, iar altele sunt folosite cu scopul de a reda controlul asupra tastaturii și *mouse*-ului sistemului de operare gazdă; de exemplu, combinația de taste *CTRL+ALT+DEL* este înlocuită de către aplicația *Microsoft Virtual PC* 2004 cu *RALT+DEL* (*RALT* - tasta *ALT* situată în partea dreaptă a tastaturii) și de către aplicația *VMWARE Workstation* 4 cu *CTRL+ALT+INS*, iar apăsarea combinației de taste *CTRL+ALT* efectuată în cadrul ferestrei unui calculator virtual găzduit de *VMWare Workstation* 4 are ca efect redarea controlului asupra tastaturii și *mouse*-ului sistemului de operare gazdă (în cadrul aplicației *Microsoft Virtual PC* 2004 acest efect se obține prin apăsarea tastei *RALT*);
- memoria internă pentru calculatoarele virtuale este constituită de o zonă din memoria internă a calculatorului gazdă și, pentru ca totul să meargă bine, dimensiunea acesteia nu poate să fie decât un anumit procent din memoria calculatorului gazdă;
- memoria externă, după cum bine se știe, este consituită în principal din *hard-disk*, *CD*-uri și *dischete*; *hard-disk*-urile pentru calculatoarele virtuale sunt constituite, în principal, de fișiere care se află pe calculatorul gazdă; aceste fișiere nu au, după cum ar fi normal, dimensiunea pe care o raportează sistemul de operare de pe calculatorul virtual, ci acestea sunt redimensionate dinamic în funcție de cerințele și necesitățile sistemului virtual; celelalte dispozitive virtuale (unități *CD-ROM*, unități de dischetă vor fi prezentate mai detaliat în secțiunea următoare;
- placa de rețea pentru sistemul de operare virtual este în întregime virtuală și există mai multe moduri în care se realizează conectarea la rețeaua la care este conectat calculatorul gazdă:
  - se poate întâmpla să nu se dorească conectarea la întreaga rețea prin intermediul plăcii de rețea virtuală, ci doar la sistemul gazdă, astfel mașinile virtuale mai instalează o placă de rețea virtuală pe sistemul gazdă pentru a realiza această conexiune, deși, pentru comunicare există alte soluții mai bune;
  - calculatorul virtual poate fi mascat în spatele celui gazdă prin utilizarea unor tabele de translație, deci se va identifica pe rețea ca fiind cel gazdă. Nu vom intra în detalii aici, deoarece aceste lucruri țin de domeniul rețelelor de calculatoare;
  - calculatorul virtual va utiliza una din plăcile de rețea a a celui gazdă, având adresă diferită de acesta, pentru a intra în legătură cu celelalte calculatoare din rețea. În cazul în care pe calculatorul gazdă există doar o placă de rețea, aceasta va fi partajată între cele două sisteme, în acest sens existând mai multe tehnici.

Memoria externă

Pe lângă faptul că pentru calculatorul virtual *hard-disk*-urile sunt constituite din fișiere stocate pe calculatorul gazdă, aplicația *VMWare Workstation* 4 oferă utilizatorilor posibilitatea de a atașa un disc al calculatorului gazdă



celui virtual cu anumite restricții, deoarece în condiții de utilizare incorectă sau rău intenționată a acestuia se poate ajunge la pierderea de date de pe propriul sistem de calcul. Aplicația *Microsoft Virtual PC* 2004 nu oferă această posibilitate, deși aceasta ar fi foarte utilă uneori.

Un calculator virtual folosește o unitate de dischetă virtuală care poate fi conectată la unitatea de dischetă a sistemului gazdă sau poate fi legată de o imagine a unei dischete (fișier care conține datele stocate în toate sectoarele unei dischete) aflată pe calculatorul gazdă.

La fel ca unitatea de dischetă, unitățile *CD-ROM* sunt virtuale, pentru ambele aplicații, și pot fi conectate la unitățile de *CD*-uri ale sistemului gazdă sau la fișiere care conțin imagini de *CD*-uri având formatul *ISO*.

Configurare, instalare, utilizare

Pentru a putea utiliza un calculator virtual trebuie mai întâi să creăm unul. Ambele aplicații oferă un ghid de creare și configurare a unui calculator virtual.

Pe parcursul operației de creare a unui sistem virtual suntem întâmpinați cu tot felul de căsuțe de dialog în care trebuie să configurăm modalitatea de conectare a *hard-disk*-ului virtual, a *CD-ROM*-ului, a unității de dischetă, a dimensiunii memoriei *RAM* care va fi disponibilă pentru acesta și a sistemului de operare pe care îl vom instala.

După ce crearea s-a terminat putem să pornim calculatorul virtual. În continuare suntem întâmpinați de mesajele pe care le afișează *BIOS*-ul acestuia. Da, și calculatoarele virtuale au un *BIOS*, deși este virtual. În *BIOS* putem seta anumiți parametri.

După ce se derulează pe ecran toate mesajele, ne întâmpină un altul care ne spune că nu poate continua pornirea deoarece nu a găsit un sistem de operare căruia să-i dea controlul, asta în cazul în care nu s-a folosit *hard-disk*-ul calculatorului gazdă pentru sistem, ceea ce la început nu este recomandat, deoarece ar putea cauza neplăceri.

Apare întrebarea: *Cum putem să instalăm un sistem de operare pe acest calculator virtual?*

Ei bine, răspunsul la această întrebare este simplu. În eventualitatea în care avem un *CD* de instalare *bootabil* (adică un *CD* dotat cu un sistem de operare) care conține aplicația de instalare a sistemului de operare dorit, îl introducem în unitatea de *CD*-uri a calculatorului gazdă și îl conectăm, prin intermediul meniului, la unitatea de *CD*-uri a calculatorului virtual și apoi repornim sistemul folosind combinația de taste adecvată.

În cazul în care avem o imagine *bootabilă* pentru acel sistem de operare, conectăm imaginea la unitatea corespunzătoare și repornim sistemul.

După câteva minute sau zeci de minute instalarea este gata. Se poate observa că sistemul de operare de pe calculatorul virtual rulează aproape la fel de repede și de bine ca și când ar fi fost instalat direct pe calculatorul gazdă.

Performanțele aplicațiilor

În continuare, pentru ca totul să meargă și mai bine, aplicațiile conțin și pachete de *driver*-e pentru diferite sisteme de operare instalate pe calculatoare virtuale.

Aceste *driver*-e sunt pentru monitorul virtual, placa video virtuală și pentru *mouse*.

Rolul acestor *driver*-e este acela de a îmbunătăți performanțele calculatorului virtual (memoria video este manipulată altfel, realizându-se o conexiune cu sistemul de operare gazdă) și de a oferi posibilitatea de a debloca *mouse*-ul și tastatura de la calculatorul virtual în momentul în care cursorul *mouse*-ului părăsește zona care reprezintă monitorul calculatorului virtual.

Microsoft Virtual PC 2004 nu conține astfel de *driver*-e pentru sistemele de operare din familia **nix*, deoarece compania *Microsoft* a dorit să ofere suport doar pentru sistemele de operare din familia *Windows* și *OS/2*. Am observat că, în ciuda acestui fapt, dacă instalăm pe un calculator virtual, folosind aplicația *Microsoft Virtual PC* 2004, sistemele de operare *Linux Redhat* 9 și *Microsoft Windows* 98, primul sistem de operare se comportă mult mai bine în mediul virtual decât cel de-al doilea pe care am instalat și *driver*-ele oferite de *Microsoft*, din punct de vedere a calității animațiilor și a vitezei de execuție a aplicațiilor.

Aplicația *VMWare Workstation* 4 oferă *driver*-e și pentru sistemele de operare din familia **nix*.

Am observat că sistemele de operare se comportă mult mai bine și sunt mai stabile pe calculatoare virtuale găzduite de *VMWare Workstation*.

La această concluzie am ajuns testând aceleași sisteme de operare pe calculatoare virtuale găzduite de ambele aplicații, având aceeași configurație și același mod de conectare a dispozitivelor virtuale. Chiar și sistemul de operare *Microsoft Windows* 98 se comportă mai bine în cadrul aplicației *VMWare Workstation* 4.

Pe calculatorul virtual găzduit de aplicația *Microsoft Virtual PC* 2004 am instalat sistemele de operare *Microsoft Windows* 98, *Linux Redhat* 9, *Slackware Linux* 9.1 și mare ne-a fost mirarea în momentul în

care, după ce am încercat să ieșim din interfața grafică *X-Windows* de pe sistemul *Slackware*, sistemul de operare *Slackware* s-a blocat și nu mai primea nici o comandă a utilizatorului, singura "scăpare" fiind un restart "*fizic*". Acest lucru nu s-a întâmplat însă și pe calculatorul virtual găzduit de aplicația *VMWare Workstation* 4.

Sistemul de operare *Microsoft Windows* 98 se comportă foarte bine pe calculatorul virtual găzduit de aplicația *VMWare Workstation* 4, chiar dacă nu sunt instalate *driver*-ele suplimentare.

În ciuda faptului că, deși sistemul de operare *Slackware Linux* 9.1 s-a blocat pe calculatorul virtual găzduit de *Microsoft Virtual PC* 2004, am observat că această aplicație nu folosește procesorul la capacitate maximă decât dacă este cazul, sistemul de operare gazdă nepierzându-și din performanță.

Microsoft Virtual PC 2004 nu pornește nici un calculator virtual dacă în memoria *RAM* a sistemului gazdă nu este liberă o cantitate egală cu cea cu care calculatorul virtual a fost configurat, în schimb cealaltă aplicație solicită sistemului de operare să mute în memoria virtuală datele din memoria *RAM* care aparțin aplicațiilor care nu sunt critice.

Ambele aplicații oferă utilizatorului posibilitatea de a conserva starea unui sistem de operare, pentru ca mai târziu să poată fi pornit din starea în care a rămas și oferă posibilitea revenirii la o stare anterioară, în cazul în care au apărut probleme cu integritatea datelor stocate pe calculatorul virtual. O altă facilitate oferită de ambele aplicații constă în partajarea de directoare de pe sistemul gazdă cu sistemul virtual în scopul schimbului de date.

Variante ale aplicațiilor

Aplicația *Microsoft Virtual PC* 2004 nu poate fi instalată decât pe sisteme de operare din familia *Windows NT* și oferă *driver*-e pentru toate sistemele de operare din familia *Windows* și *OS/2*, în timp ce aplicația *VMWare Workstation* 4 poate fi instalată atât pe sisteme de operare din familia *Windows NT*, cât și pe sisteme de operare din familiile **nix* și *OS/2*.

Noutăți

Microsoft a anunțat faptul că vor fi disponibile *driver*-e și pentru alte sisteme de operare.

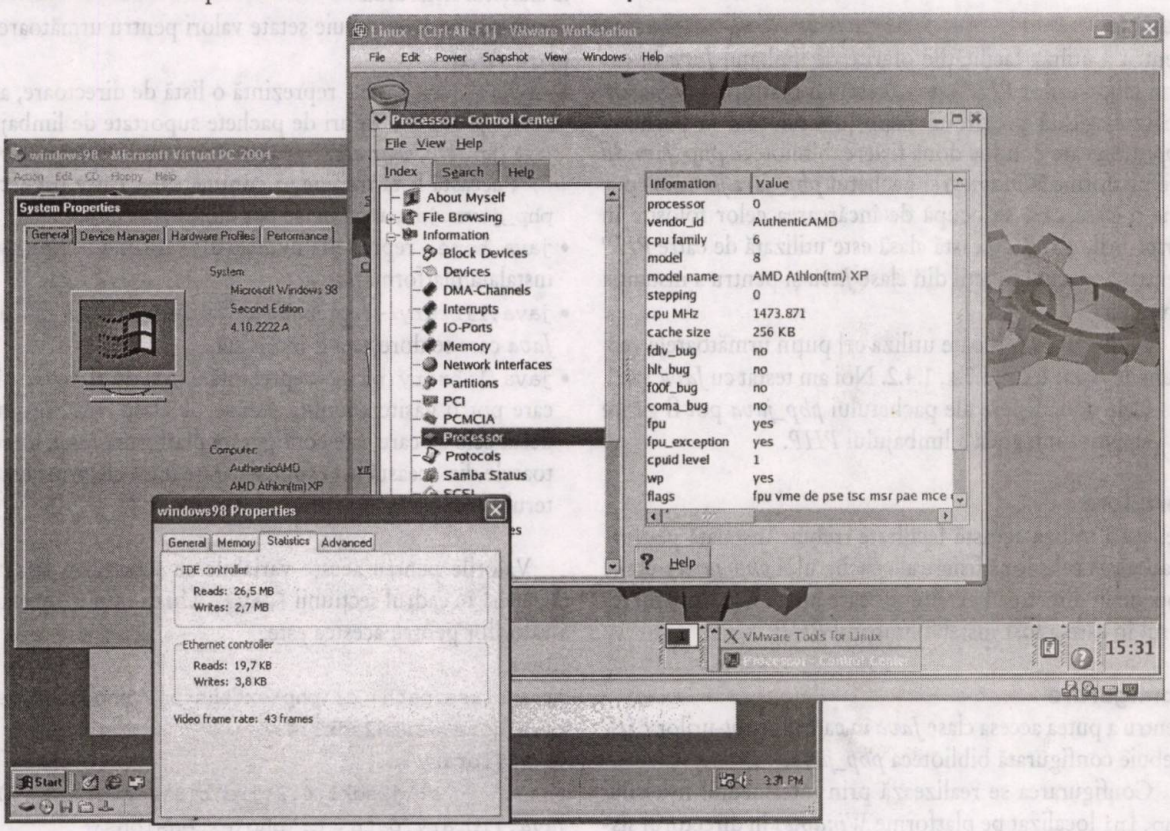
Corporația *VMWare* a anunțat că în versiunile următoare va oferi utilizatorilor posibilitatea de a crea și utiliza calculatoare virtuale pe care să ruleze sistemele de operare din familia *MacOS*.

Rămâne de văzut care vor fi performanțele acestor componente pe care cele două companii le-au anunțat.

Concluzii

Deși sistemele de operare rulează mult mai bine dacă sunt instalate pe un calculator fizic, uneori sunt necesare, pentru prelucrare, date aflate pe un sistem de operare altul decât cel care rulează. În acest scop cea mai bună alegere, dacă avem mai multe sisteme de operare instalate pe sistemul fizic, este *VMWare Workstation* 4, deoarece putem atașa *hard-disk*-ul curent la calculatorul virtual.

Informații suplimentare legate de aceste două produse pot fi găsite la adresele **http://www.microsoft.com**, respectiv **http://www.vmware.com**.



Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

Limbajul PHP oferă realizatorilor de pagini web dinamice o multitudine de clase și funcții. În acest articol vom prezenta felul în care, într-un script scris în PHP 4, se pot utiliza clase Java și felul în care se pot realiza imagini dinamic.

Interoperabilitate PHP/Java

Un aspect important al limbajului *PHP* este acela că realizatorii de pagini *web* pot folosi în *script*-uri o mare parte a facilităților oferite de limbajul *Java* prin intermediul unei biblioteci de funcții numită ***php_java***.

Această facilitate îmbunătățește foarte mult limbajul *PHP*, aducându-i acestuia o foarte mare flexibilitate.

În continuare vom prezenta modul în care se pot instanția și utiliza clase *Java* în cadrul *script*-urilor *PHP*.

Cerințe

Pentru a utiliza facilitățile oferite de limbajul *Java* în cadrul *script*-urilor *PHP* este necesară o platformă *Java* care poate fi găsită pe site-ul **http://java.sun.com** și pachetul *php_java* care conține două fișiere: biblioteca ***php_java.dll*** (pe platforme *Windows*) și pachetul ***php_java.jar*** care conține o clasă care se ocupă de încărcarea celor folosite în *script*-urile *PHP*. Această clasă este utilizată de către *PHP* pentru a executa funcții din clase *Java* și pentru a instanția clase noi.

Limbajul *PHP* poate utiliza cel puțin următoarele versiuni de *Java*: 1.2.x, 1.3.x, 1.4.2. Noi am testat cu *Java* 1.4.2.

Cele două fișiere ale pachetului *php_java* pot fi găsite în versiunea integrală a limbajului *PHP*.

Instalare

Pentru a utiliza această facilitate trebuie instalată platforma *Java* și cele două fișiere ale pachetului *php_java* trebuie copiate în directorul *extensions* care poate fi găsit în directorul în care a fost instalat limbajul *PHP*.

Configurare

Pentru a putea accesa clase *Java* în cadrul *script*-urilor *PHP* trebuie configurată biblioteca *php_java*.

Configurarea se realizează prin intermediul fișierului `php.ini` localizat pe platforme *Windows* în directorul sistemului de operare, și anume directorul care constituie valoarea variabilei sistem `%windir%`.

În acest fișier trebuie adăugată valoarea `php_java.dll` variabilei `extension`. Există mai multe posibilități pentru a face acest lucru. Una dintre ele este eliminarea caracterului " ; " de la începutul liniei:

```
;extension=php_java.dll,
```

dacă o astfel de linie există în fișierul `php.ini`, iar o altă variantă este adăugarea liniei: `extension=php_java.dll` la sfârșitul fișierului.

În continuare trebuie setate valori pentru următoarele patru variabile:

- `java.class.path` - reprezintă o listă de directoare, arhive *zip* sau alte tipuri de pachete suportate de limbajul java (*jar*, *ejb*, *war* etc.) separate între ele prin caracterul " ; "; această listă trebuie să conțină calea către pachetul `php_java.jar`, altfel nu se pot utiliza clase *Java*;
- `java.home` - reprezintă numele directorului în care este instalată platforma *Java*;
- `java.library` - reprezintă calea către mașina virtuală *Java* care se dorește a fi încărcată;
- `java.library.path` - reprezintă o listă de directoare în care pot fi găsite anumite pahete de clase *Java*, în plus decât cele cu care este configurată platforma *Java*; directoarele din această listă sunt separate între ele prin caracterul " ; ".

Valorile pentru aceste variabile se setează în fișierul `php.ini` în cadrul secțiunii `[Java]`. Un exemplu de setare a valorilor pentru acestea este:

```
[Java]
java.class.path = c:\php\extensions\php_java.jar
java.home = c:\j2sdk1.4.2
java.library =
        c:\j2sdk1.4.2\jre\bin\server\jvm.dll
java.library.path = c:\php\extensions\
```

Cu aceste setări făcute, mașina virtuală *Java* poate fi încărcată și clasele pot fi utilizate în cadrul *script*-urilor *PHP*.

Utilizare

Clasele *Java* se instanțiază în *PHP* folosind constructorul clasei *Java* explicat în secțiunea următoare.

Metodele și proprietățile statice sau dinamice ale claselor instanțiate se accesează folosind operatorul de indirectare "->". De exemplu, dacă avem o variabilă `$clasa` căreia i-a fost atașată ca valoare o clasă *Java*, atunci pentru a afișa reprezentarea textuală a acesteia trebuie să utilizăm metoda `toString()` (metodă specifică tuturor claselor *Java*) vom utiliza o instrucțiune de forma:

```
echo $clasa->toString();.
```

Clasa Java

Această clasă se folosește pentru a instanția o clasă a limbajului *Java* și reprezintă, de fapt, o clasă derivată din clasa *java.lang.Object* a limbajului *Java*.

Constructorul acesteia primește mai mulți parametri, dintre care primul este un șir de caractere care reprezintă numele clasei *Java* care se instanțiază, iar restul de parametri reprezintă parametrii pentru care se va apela constructorul clasei care se instanțiază.

Rezultatul apelării constructorului clasei *Java* constă într-o clasă a acestui limbaj care are tipul dat ca parametru constructorului.

În cazul în care nu se reușește instanțierea unei clase a limbajului *Java* este generată o excepție *Java*, iar pentru a verifica dacă a fost generată o astfel de excepție se folosește funcția *java_last_exception_get* care este prezentată în secțiunea imediat următoare.

În continuare vom prezenta un exemplu de utilizare a clasei *Java*:

```
<HTML>
 <HEAD>
  <TITLE>
   Java &amp; PHP
   </TITLE>
 </HEAD>
 <BODY>
    <?PHP
      $system = new Java("java.lang.System");
      echo "Versiunea Java este: ";
      echo $system->
          getProperty("java.version")."<BR>";
      echo "Sistem de operare: ";
      echo $system->getProperty("os.name")
                                        ."<BR>";
      echo "Versiunea: ";
      echo $system->getProperty("os.version")
                                        ."<BR>";
    ?>
  </BODY>
</HTML>
```

În figura alăturată poate fi observat efectul execuției acestui cod sursă care folosește constructorul *Java* pentru a obține o instanță a clasei statice *java.lang.System*.

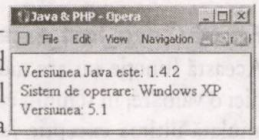


Figura 1: Clasa Java

Funcția java_last_exception_get

Această funcție returnează o clasă *Java* derivată din clasa *java.lang.Exception* care reprezintă ultima excepție care a fost generată sau valoarea logică *false* dacă nu a fost generată nici o excepție de la începutul execuței *script*-ului sau de la ultimul apel al funcției *java_last_exception_clear*.

Funcția *java_last_exception_get* nu are parametri.

În continuare prezentăm un exemplu de utilizare a acestei funcții și în figura 2 poate fi observat efectul execuției codului:

```
<HTML>
  <HEAD>
    <TITLE>
      Conversie
    </TITLE>
  </HEAD>
  <BODY>
    <FORM action = "exceptie.php" method = post>
      Introduceti un șir de caractere:
      <INPUT type = text name = "sir">
      <INPUT type = submit value = "Verificare">
    </FORM>
    <?PHP
      if (isset($_POST["sir"])){
        echo "<HR>";
        $int = new Java("java.lang.Integer");
        $sir = $_POST["sir"];
        $i = $int->parseInt($sir);
        $error = java_last_exception_get();
        if ($error){
          echo "Șirul \"".$sir.
                    "\" nu este număr întreg.";
          java_last_exception_clear();
        }
        else{
          echo "Șirul \"".$i.
                    "\" este număr întreg.";
        }
      }
    ?>
  </BODY>
</HTML>
```

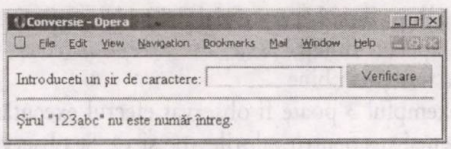


Figura 2: Funcția java_last_exception_get

Funcția java_last_exception_clear

Această funcție nu are nici un parametru și nu returnează nici o valoare, în schimb efectul aplicării ei este acela că este anulată ultima excepție *Java* care a fost generată de către mașina virtuală *Java* în timpul executării de cod.

Apelul acestei funcții este necesar, deoarece, în cazul în care a fost generată o excepție și nu a fost tratată, se poate întâmpla ca după apelul unei funcții să tratăm excepția greșită. De exemplu, în cazul următorului cod se tratează excepția greșită dacă atribuim lui *a* valoarea `abc` și lui *b* valoarea `12`, iar rezultatul nu este cel dorit:

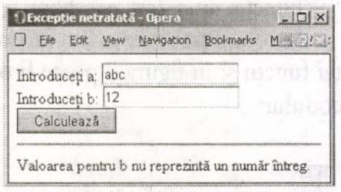


Figura 3: Excepție netratată

```
<HTML>
  <HEAD>
    <TITLE>
      Excepție
          netratată
    </TITLE>
  </HEAD>
  <BODY>
    <FORM action = "exceptie2.php" method = post>
      Introduceți a:
      <INPUT type = text name = "nr1">
      <BR>
      Introduceți b:
      <INPUT type = text name = "nr2">
      <BR>
      <INPUT type = submit value = "Calculează">
    </FORM>
    <?PHP
      if ((isset($_POST["nr1"])) &&
                        (isset($_POST["nr2"]))){
        echo "<HR>";
        $int = new Java("java.lang.Integer");
        $a = $int->parseInt($_POST["nr1"]);
        $b = $int->parseInt($_POST["nr2"]);
        $error = java_last_exception_get();
        if ($error){
          echo "Valoarea pentru b nu reprezintă
                                  un număr întreg.";
          java_last_exception_clear();
        }
        else {
      echo "b^2 = ".$b*$b;
        }
      }
    ?>
  </BODY>
</HTML>
```

Dacă după linia în care variabilei `$a` i se atribuie valoarea `abc` apelăm funcția *java_last_exception_clear*, atunci codul va funcționa bine.

În exemplul 3 poate fi observat efectul executării codului de mai sus pentru valorile `abc` și `12`. Se observă că în loc să se afișeze `b^2 = 144`, se afișează o eroare.

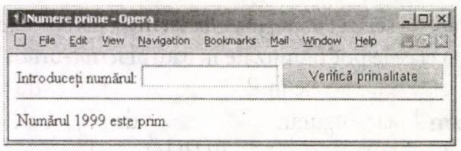


Figura 4: Tratarea corectă a excepțiilor

În continuare prezentăm un exemplu de utilizare corectă a celor două funcții de manipulare a erorilor generate de programele care rulează pe mașina virtuală *Java*, iar rezultatul execuției acestuia poate fi observat în figura 4.

```
<HTML>
  <HEAD>
    <TITLE>
      Numere prime
    </TITLE>
  </HEAD>
  <BODY>
    <FORM action = "prim.php" method = post>
    Introduceți numărul:
    <INPUT type = text name = "number">
    <INPUT type = submit value = "Verifică
                                  primalitate">
    </FORM>
    <?PHP
      if (isset($_POST["number"])){
        echo "<HR>";
        $bint = new Java("java.math.BigInteger",
                                  $_POST["number"]);
        $error = java_last_exception_get();
        if ($error){
          echo "Număr invalid.";
          java_last_exception_clear();
        }
        else{
          echo "Numărul ".$_POST["number"];
          $p=20;
          $p=(int)$p;
          if ($bint->isProbablePrime($p))
            echo " este prim.";
          else
            echo " nu este prim.";
        }
      }
    ?>
  </BODY>
</HTML>
```

Importanța librăriei php_java

Pentru programatorii de pagini *web* care cunosc atât limbajul *Java*, cât și limbajul *PHP*, prezența acestei librării are o foarte mare importanță.

Limbajul *Java* are, pe lângă limbajul *PHP*, o serie de facilități care pot fi folosite cu succes în cadrul paginilor *web*.

Java oferă clase pentru prelucrarea diferitelor tipuri de fișiere, lucru care se face mult mai ușor decât în *PHP*, clase

pentru acces la resurse aflate la distanță. În plus, funcțiile scrise în *Java* sunt executate de către procesor mult mai repede decât cele scrise în *PHP*, iar utilizarea codului *Java* oferă o mai mare siguranță creatorilor de pagini *web* dinamice.

Datorită faptului că limbajul *PHP* nu este puternic tipizat, variabilor fiindu-le atașate un tip în mod dinamic, în cadrul scripturilor *PHP* nu se pot utiliza toate facilitățile oferite de platforma *Java*, cum ar fi utilizarea conversiilor explicite de tip, ceea ce îngreunează puțin lucrul cu clase *Java* în paginile *PHP*, fiind necesare clase în *Java* suplimentare, realizate de către programator, care să facă exact ceea ce trebuie.

Singurele conversii explicite de tip care se pot face sunt conversiile între tipurile de bază comune ale celor două limbaje.

Dacă realizați clase pe care doriți să le instanțiați în cadrul *script*-urilor *PHP*, să nu uitați să configurați fișierul `php.ini` corespunzător.

Prelucrarea dinamică a imaginilor

Un alt aspect important al limbajului *PHP* este acela că oferă programatorilor posibilitatea de a crea și publica în cadrul paginilor *web* imagini dinamice.

În cazul în care vă întrebați în ce condiții sunt folosite imaginile generate dinamic vă pot da un exemplu simplu, și anume, gândiți-vă la o aplicație care monitorizează accesările unei pagini *web* și afișează rezultatul sub forma unei imagini.

Pentru a profita de această facilitate în limbajul *PHP* trebuie activată mai întâi biblioteca `php_gd2`. Acest lucru se face prin adăugarea următoarei linii în fișierul de configurare *php.ini* (exemplul de configurare este pentru sistemele de operare din familia *Windows*):

```
extension = php_gd2.dll
```

În acest moment sunt disponibile funcțiile de prelucrare și generare dinamică de imagini.

Această librărie permite și adăugarea de text în cadrul imaginilor generate și oferă suport pentru imagini în mai mai multe formate.

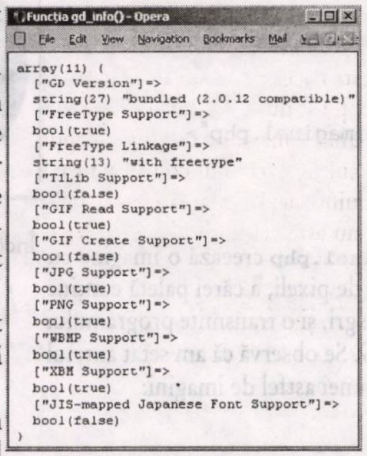
Pentru a obține imagini de calitate foarte bună, această bibliotecă oferă suport integrat pentru bibliotecile *FreeType* 1, *FreeType* 2 și *T1Lib*, care au ca scop încărcarea și desenarea *font*-urilor independent de platformă, însă nu vom discuta în cadrul acestui articol despre aceste trei biblioteci, ci numai vom aminti funcțiile oferite de limbajul *PHP* pentru accesarea lor.

În continuare vom prezenta cele mai importante funcții, oferite de limbajul *PHP*, necesare pentru prelucrarea imaginilor.

Formate de imagini suportate

Pentru a putea afla ce formate de imagini suportă biblioteca instalată (în funcție de versiunea *PHP*) se poate executa funcția `gd_info()`.

În continuare vă prezentăm un exemplu de utilizare a acestei funcții și facilitățile oferite de biblioteca `php_gd2` și în figura 5 poate fi observat efectul execuției codului.




```
<HTML>
  <TITLE>
    Funcția
          gd_info()
  </TITLE>
  <BODY>
    <PRE>
    <?PHP
      var_dump
        (gd_info());
    ?>
    </PRE>
  </BODY>
</HTML>
```

Figura 5: Funcția gd_info

Rezultatul returnat de această funcție îl reprezintă un vector cu mai multe componente.

Se poate observa că printre aceste componente se află versiunea bibliotecii și facilitățile oferite.

Formatele de imagini suportate de biblioteca *php_gd2* sunt *GIF*, *JPG*, *PNG*, *WBMP* și *XBM* (*XPM*). Se observă că biblioteca folosită de noi nu permite salvarea imaginilor în format *GIF*.

Crearea dinamică a imaginilor

Pentru a crea imagini în mod dinamic avem la dispoziție două opțiuni. Putem crea o imagine care este bazată pe o paletă de culori (pentru acest tip de imagini avem la dispoziție doar 256 de culori cu care putem opera) sau putem crea o imagine cu număr neprecizat de culori (*true color*).

Funcțiile cu ajutorul cărora se pot crea imagini sunt:

- `imagecreate` - această funcție creează o imagine goală care este bazată pe o paletă de culori; primește ca parametri două numere întregi care reprezintă dimensiunile imaginii (lățimea și înălțimea) și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea nou creată;
- `imagecreatetruecolor` - această funcție creează o imagine care are fundalul negru și număr nedefinit de culori; primește ca parametri două numere întregi care reprezintă dimensiunile imaginii (lățimea și înălțimea) și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea nou creată.

Trebuie avut în vedere, că pentru a afișa o imagine dinamică pe o pagină *web* trebuie realizată o pagină care are un element **`IMG`** al cărui atribut **`src`** să indice calea către *script*-ul *PHP* care creează imaginea.

În continuare vă prezentăm un exemplu de creare a unei imagini care se bazează pe o paletă de culori împreună cu o pagină web simplă în cadrul căreia este afișată imaginea.

Pagina *web* este:

```
<HTML>
  <TITLE>
    Afișarea imaginii dinamice
  </TITLE>
  <BODY>
    <IMG src = "creareimagine1.php">
  </BODY>
</HTML>
```

Script-ul **creareimagine1.php** creează o imagine cu lățimea și înălțimea de 100 de pixeli, a cărei paletă conține o singură culoare, și anume gri, și o transmite programului de navigare în format *PNG*. Se observă că am setat antetul corespunzător transmisiei unei astfel de imagini.

```
<?PHP
  header("Content-type: image/png");
  $im = imagecreate(100, 100);
  $fundal = imagecolorallocate($im,64,64,64);
  imagepng($im);
  imagedestroy($im);
?>
```

Figura 6: Funcția imagecreate

În figura alăturată poate fi observat efectul încărcării paginii *web* în fereastra programului de navigare.

Încărcarea imaginilor

Biblioteca permite, de asemenea, citirea de imagini aflate la o anumită locație și încărcarea acestora în memorie pentru prelucrare și, eventual, publicare.

Funcțiile care încarcă imagini deja existente sunt:

- `imagecreatefromgif` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *GIF* încărcată;
- `imagecreatefromjpeg` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *JPG* încărcată;
- `imagecreatefrompng` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *PNG* încărcată;
- `imagecreatefromwbmp` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *WBMP* încărcată;
- `imagecreatefromxbm` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *XBM* încărcată;
- `imagecreatefromxpm` - această funcție primește ca parametru o cale spre un fișier, care se poate afla și la distanță, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea de tipul *XPM* încărcată;

Figura 7: Încărcarea unei imagini

- `imagecreatefromstring` - această funcție primește ca parametru un șir de caractere care conține reprezentarea binară a unei imagini, și returnează o variabilă de tipul `resource` care reprezintă identificatorul de acces la imaginea încărcată.

În continuare vă prezentăm un exemplu de utlizare a acestei funcții:

```
<?PHP
  header("Content-type: image/png");
  $im = imagecreatefromgif("images/ginfo.gif");
  imagepng($im);
  imagedestroy($im);
?>
```

În figura 7 poate fi observat efectul execuției acestui cod dintr-o pagină *web*.

În continuare vă prezentăm câteva clase de funcții care pot fi utilizate la prelucrarea imaginilor.

Desenarea pe imagini

Pentru a desena pe o imagine creată dinamic sau încărcată dintr-un fișier avem la dispoziție mai multe funcții:

- `imagearc` - desenează pe o imagine un arc de elipsă; primește ca parametri identificatorul de acces la imaginea pe care se va desena, coordonata *x* a centrului elipsei, coordonata *y* a centrului elipsei, lungimea și înălțimea elipsei, unghiurile de start și de stop ale arcului și culoarea cu care se va desena; desenarea se face în sens trigonometric;
- `imagefilledarc` - desenează pe o imagine un arc de elipsă haşurat; are aceeași parametri cu funcția anterioară la care se mai adaugă unul care reprezintă stilul haşurii și care poate avea valorile: `IMG_ARC_PIE`, `IMG_ARC_CHORD`, `IMG_ARC_NOFILL`, `IMG_ARC_EDGED`;
- `imageline` - desenează o linie pe o imagine; primește ca parametri identificatorul de acces al imaginii pe care se va desena, coordonata *x* a primului capăt al liniei, coordonata *y* a primului capăt, coordonata *x* a celui de-al doilea capăt, coordonata *y* a celui de-al doilea capăt și culoarea cu care se va desena linia;
- `imagedashedline` - desenează o linie punctată pe o imagine; are aceeași parametri ca funcția anterioară și va folosi stilul implicit sau cel setat de utilizator;

- `imagesetthickness` - setează grosimea cu care se vor trasa liniile pentru o anumită imagine; primește ca parametri identificatorul de acces al imaginii și un număr întreg care reprezintă grosimea cu care se vor desena liniile pe imagine;
- `imageellipse` - desenează o elipsă pe o imagine; primește ca parametri identificatorul de acces al imaginii pe care se va desena, coordonata *x* a centrului elipsei, coordonata *y* a centrului elipsei, lungimea și înălțimea elipsei și culoarea cu care se va desena;
- `imagefilledellipse` - desenează o elipsă plină; are aceeași parametri cu funcția anterioară;
- `imagepolygon` - desenează un poligon pe o imagine; primește ca parametri identificatorul de acces al imaginii pe care se va desena poligonul, un tablou ale cărui elemente sunt numere întregi și reprezintă coordonatele punctelor poligonului care se va desena, numărul de puncte ale poligonului și culoarea cu care se va desena; tabloul cu punctele poligonului conține pe pozițiile pare coordonatele *x* ale punctelor, iar pe pozițiile impare coordonatele *y* ale punctelor;
- `imagefilledpolygon` - desenează un poligon plin; are aceeași parametri cu funcția anterioară;
- `imagerectangle` - desenează un dreptunghi pe o imagine; primește ca parametri identificatorul de acces al imaginii pe care se va face desena, coordonata *x* a colțului din partea stânga-sus a dreptunghiului, coordonata *y* a colțului din partea stânga-sus a dreptunghiului, coordonata *x* a colțului din partea dreapta-jos a dreptunghiului, coordonata *y* a colțului din partea dreapta-jos a dreptunghiului și culoarea cu care se va desena;
- `imagefilledrectangle` - desenează un dreptunghi plin; are aceeași parametri cu funcția anterioară;
- `imagefill` - umple cu o culoare o zonă de pe imagine care are aceeași culoare cu cea a punctului care se află la coordonatele specificate ca parametru; primește ca parametri identificatorul de acces al imaginii pe care se va face umplerea, coordonata *x* a unui punct din zona care se va umple cu culoare, coordonata *y* a punctului și culoarea cu care se va umple;
- `imagefilltoborder` - umple cu o culoare o zonă de pe imagine care este mărginită de o culoare specificată ca parametru; primește ca parametri identificatorul de acces al imaginii pe care se va face umplerea, coordonata *x* a unui punct din zona care se va umple cu culoare, coordonata *y* a punctului, culoarea punctelor cu care este mărginită zona și culoarea cu care se va umple;
- `imagesetpixel` - schimbă culoarea unui punct de pe o imagine; primește ca parametri identificatorul de acces al imaginii, coordonata *x* a punctului a cărui culoare se va schimba, coordonata *y* a punctului și noua culoarea a acestuia;
- `imagecolorat` - returnează culoarea unui anumit punct de pe imagine; primește ca parametri identificatorul de acces al imaginii, coordonata *x* a punctului și coordonata *y* a punctului;
- `imagecolorallocate` - primește ca parametri identificatorul de acces al unei imagini, o valoare pentru componenta *roșu* a unei culori, o valoare pentru componenta *verde* a unei culori și o valoare pentru componenta *albastru* a unei culori și returnează un număr întreg care reprezintă culoarea având cele trei componente; în cazul în care imaginea se bazează pe o paletă de culori, atunci se returnează indicele culorii din paletă; dacă în paleta de culori nu există o culoarea care să aibă toate cele trei componente, atunci ea este adăugată la paleta de culori a imaginii;
- `imagecolordeallocate` - primește ca parametri identificatorul de acces al unei imagini și o culoare; în cazul în care imaginea specificată ca parametru se bazează pe o paletă de culori, atunci culoarea este eliminată din paletă.

În continuare prezentăm un exemplu care folosește o parte din funcțiile prezentate anterior. În figura 8 poate fi observat efectul execuției codului din cadrul unei pagini *web*:

```
<?PHP
   header("Content-type: image/png");
   $im = imagecreatefromjpeg("image/ginfo.jpg");

  $red = imagecolorallocate($im,255,0,0);
   $blue = imagecolorallocate($im,0,0,255);
   $green = imagecolorallocate($im,0,255,0);
   $yellow = imagecolorallocate($im,255,255,0);
   $cyan = imagecolorallocate($im,0,255,255);
   $magenta = imagecolorallocate($im,255,0,255);

   imagefill($im,229,229,$cyan);
   imagefill($im,30,100,$magenta);
   imagefill($im,80,160,$yellow);

   imagefilledrectangle($im,0,0,50,100,$red);
   imagefilledellipse($im,100,100,50,50,
                                    $green);
   imagesetthickness($im,3);
   $poly = array(10,10,200,50,225,100,150,200,
                                  100,75,50,50);
   imagefilledpolygon($im,$poly,6,$blue);
   imageline($im,0,229,229,0,$red);
   imagepng($im);
   imagedestroy($im);
?>
```

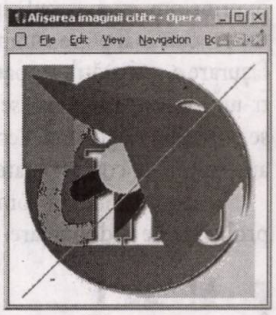


Figura 8: Exemplu prelucrare

Va urma...

În articolul următor vom continua prezentarea funcțiilor referitoare la prelucrarea imaginilor și vom începe prezentarea modului de utilizare a bazelor de date *MySQL* în interiorul *script*-urilor *PHP*.

Algoritmi de ORDONANȚARE

Ilie Vieru

Ordonanțarea este o problemă de optimizare specifică sistemelor dinamice în care apar o serie de dereglări care determină un comportament necorespunzător al sistemului față de obiectivul acestuia. În cele ce urmează vom prezenta trei probleme de ordonanțare împreună cu algoritmii de rezolvare.

În prima problemă de ordonanțare tratată în acest articol se folosește *diagrama Gantt* și se pune în evidență algoritmul corespunzător. Apoi vom rezolva problema minimizării timpului total pentru prelucrarea a n piese pe două mașini (algoritmul lui *Johnson*). În ultima problemă se minimizează suma penalizărilor pentru neîncadrarea în grafic (algoritmul lui *Moore*).

Ordonanțarea este o noțiune întâlnită în diverse domenii de activitate, cu precădere în procesele de organizare și planificare a fluxului tehnologic (operațiilor) dintr-o unitate de producție, de planificare și corelare a activităților la o societate de construcții-montaj sau în oricare altă unitate în care se dorește organizarea muncii, disciplină, încadrare în termene, îmbunătățire a performanțelor etc.

Prin ordonanțare se urmărește programarea activităților critice (care nu pot fi întârziate și nici devansate), precum și determinarea rezervelor de timp pentru activitățile mai puțin critice.

Folosind teoria grafurilor, aceste probleme se rezolvă, în principiu, cu ajutorul algoritmul *Bellman-Kalaba*. Pentru procese mai puțin complexe, pentru care ordinea de desfășurare a activităților poate fi reprezentată și urmărită într-un grafic de timp care se mai numește și ***diagrama Gantt***, se pot utiliza algoritmi elementari, dar foarte eficienți (care au ordinul de complexitate $O(n \cdot \ln n)$).

În cele ce urmează vom identifica algoritmii pentru trei probleme de ordonanțare.

Problema 1

Într-un atelier se execută n piese la două mașini M_1 și M_2; fiecare piesă se prelucrează mai întâi pe prima mașina M_1, apoi pe mașina M_2. O mașină nu poate prelucra simultan mai multe piese. Timpii necesari prelucrării pieselor pentru fiecare mașină sunt specificați într-un tabel de forma:

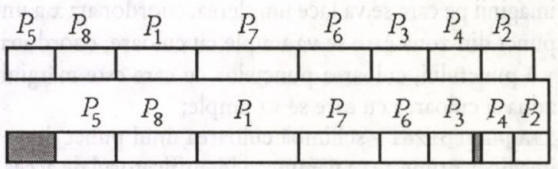
	P_1	P_2	P_3	...	P_n
M_1	a_1	a_2	a_3	...	a_n
M_2	b_1	b_2	b_3	...	b_n

unde a_i reprezintă timpul necesar pentru a realiza operația de prelucrare a piesei i pe mașina M_1, iar b_i reprezintă timpul necesar pentru a realiza operația de prelucrare a piesei i pe mașina M_2, $i \in \{1, 2, \ldots, n\}$.

Presupunând că piesele se prelucrează în această ordine, să se determine timpul total de lucru pentru fiecare mașină, precum și timpul de așteptare pentru mașina M_2.

Rezolvare

Vom începe descrierea rezolvării acestei probleme prezentând *diagrama Gantt*:



pentru exemplul:

P_5	P_8	P_1	P_7	P_6	P_3	P_4	P_2
12	15	20	25	18	14	12	10
15	20	25	20	12	11	10	8

Observăm că timpul total de lucru pentru mașina M_1 nu conține timpi de așteptare, deci toate operațiile sunt cri-

tice; mașina M_2 așteaptă 12 unități de timp, necesare și suficiente pentru terminarea prelucrării primei piese pe mașina M_1, iar după realizarea operației pentru piesa P_3, mașina M_2 rămâne din nou fără piese pentru lucru și trebuie să mai aștepte o unitate de timp, deci $T_S = 13$. Notăm cu t_1 și t_2 timpul necesar prelucrării pieselor pe mașina M_1, respectiv mașina M_2, iar cu t_3 timpul de așteptare (stagnare) pentru mașina M_2.

Cu aceste notații avem următorul algoritm:

```
t1 ← a1                                          (1)
t2 ← a1
i ← 1
j ← 0
repetă
   dacă t1 = t2 atunci                           (2)
   i ← i + 1
   j ← j + 1
   t1 ← t1 + ai
   t2 ← t2 + bj
   altfel
      dacă t1 < t2 atunci                        (3)
         i ← i + 1
         t1 ← t1 + ai
      altfel
         dacă j < i - 1 atunci                   (4)
            j ← j + 1
            t2 ← t2 + bj
         altfel                                  (5)
            t3 ← t3 + t1 - t2
            t2 ← t1
         sfârșit dacă
      sfârșit dacă
   sfârșit dacă
până când i = n                                  (6)
```

Observații:

(1) La început M_1 lucrează la prima piesă, iar M_2 așteaptă terminarea acestei operații.

(2) Dacă cele două mașini termină simultan prelucrarea piesei i, respectiv j ($i \neq j$), ambele trec la prelucrarea pieselor următoare.

(3) Dacă M_1 termină prelucrarea piesei i, iar M_2 lucrează încă la piesa j, atunci M_1 trece la prelucrarea piesei următoare.

(4) Dacă M_2 termină piesa j și mai are piese (prelucrate de M_1) trece la prelucrarea piesei următoare.

(5) Dacă M_2 nu mai are piese, așteaptă să termine M_1 piesa i la care lucrează.

(6) În acest moment M_1 a terminat lucrul, iar M_2 mai are de prelucrat piesele: $j + 1, j + 2, \ldots, n$.

Problema 2

În condițiile problemei 1, să se ordonanțeze operațiile de prelucrare a pieselor pe mașinile M_1 și M_2, astfel încât timpul total de lucru pe mașina M_2 să fie minim.

Rezolvare

Această problemă se rezolvă cu ajutorul algoritmului lui *Johnson* care conține următorii trei pași:

- alegem valoarea minimă dintre vectorii a și b care presupunem că se găsește în poziția s;
- dacă minimul este a_s, vom începe prelucrarea pe mașina M_1 cu piesa având numărul de ordine s, în caz contrar, vom termina lucrarea cu piesa de pe poziția s;
- eliminăm a_s și b_s și repetăm acești trei pași până la construirea succesiunii optime de prelucrare a celor n piese.

Dacă notăm cu l_s și l_d capetele curente ale intervalului de sortat și $r = 1$ sau $r = 2$, după cum minimul s-a găsit în șirul a, respectiv b, algoritmul se poate reprezenta astfel:

```
ls ← 1
ld ← n
pentru k = 1, n - 1 execută:
   dacă a[ls] < b[ls] atunci
      min ← a[ls]
      r ← 1
   altfel
      min ← b[ls]
      r ← 2
   sfârșit dacă
   pmin ← ls
   pentru i = ls + 1, ld execută:
      dacă ai < min atunci
         min ← ai
         prim = i
         r = 1
      sfârșit dacă
      dacă bi < min atunci
         min = bi
         prim = i
         r = 2
      sfârșit dacă
   sfârșit pentru
   dacă r = 1 atunci
      aux ← a[pmin]
      a[pmin] ← a[ls]
      a[ls] ← aux
      aux ← b[pmin]
      b[pmin] ← b[ls]
      b[ls] ← aux
      ls ← ls + 1
   altfel
      aux ← a[pmin]
      a[pmin] ← a[ld]
      a[ld] ← aux
      aux ← b[pmin]
      b[pmin] ← b[ld]
      b[ld] ← aux
      ld ← ld - 1
   sfârșit dacă
sfârșit pentru
```

Cunoscând ordinea de prelucrare, putem aplica algoritmul de la problema anterioară și vom obține rezultatul căutat. Exemplul prezentat la problema 1 reprezintă ordinea optimizată a lucrărilor pe cele două mașini.

Problema 3

Să presupunem că o mașină M trebuie să prelucreze n piese. Pentru fiecare piesă se cunosc: timpul de execuție a_j, timpul limită de predare b_j (raportat la momentul de începere a lucrului $t = 0$), precum și penalizarea c_j în cazul în care predarea nu se face la termen, toate fiind numere reale strict pozitive. Să se determine o ordine de prelucrare a pieselor care să minimizeze suma penalizărilor corespunzătoare pieselor realizate după termenele limită stabilite.

Observație: Dacă toate valorile a_j și c_j sunt numere întregi, iar valorile b_j sunt egale, problema este *NP- completă* [1]. Totuși, în cazul în care toate valorile c_j sunt egale, *J. M. Moore* a propus un algoritm cu timp de lucru polinomial. Acest algoritm se poate extinde la un caz puțin mai general, și anume pentru oricare două piese i și j să se verifice condiția: $a_i < a_j \Rightarrow c_i \geq c_j$, adică, penalizările sunt considerate invers proporționale cu timpii de execuție.

Rezolvare

În rezolvarea acestei probleme se pornește de la observația simplă că întotdeauna există o succesiune optimă pentru care toate lucrările neîntârziate se execută primele, în ordinea crescătoare a timpilor limită, urmată de cele întârziate care se pot executa într-o ordine arbitrară. Va trebui, deci, să determinăm un algoritm pentru selectarea optimă a lucrărilor (pieselor) care se vor realiza fără întârzieri. Conform [1] se introduce relația de ordine totală pe mulțimea lucrărilor:

$$i \; p \; j \Leftrightarrow \begin{cases} a_i > a_j, \text{sau} \\ a_i = a_j \text{ și } c_i < c_j, \text{sau} \\ a_i = a_j, c_i < c_j \text{ și } i < j \end{cases}.$$

Algoritmul folosit constă din următorii doi pași:

- se sortează lucrările în ordinea crescătoare a timpilor limită b;
- se construiesc recurent mulțimile lucrărilor j-optime astfel:

$$S_1 = \{1\}$$

$$S_{j+1} = \begin{cases} S_j \cup \{j+1\}, \text{dacă} p(S_j) + a_{j+1} \leq b_{j+1}, \\ (S_j \cup \{j+1\}) - \{k\}, \text{în caz contrar} \end{cases}$$

unde k este elementul minim relativ la relația p din mulțimea $S_j \cup \{j+1\}$, iar $p(S)$ reprezintă suma timpilor de efectuare a lucrărilor din S. Implementarea relației de ordine totală " p " pe care o vom nota *ordn*(*i*, *j*) presupune:

```
dacă a_si > a_sj atunci
  returnează i
sfârșit dacă
dacă a_si < a_sj atunci
  returnează j
sfârșit dacă
dacă c_si < c_sj atunci
  returnează i
sfârșit dacă
dacă c_si > c_sj atunci
  returnează j
sfârșit dacă
dacă i < j atunci
  returnează i
sfârșit dacă
returnează j
```

Cu aceasta, algoritmul pentru construirea mulțimilor j-optime devine:

```
p ← 1
s_p ← 1
S_a ← a_sp
S_b ← b_p
pentru j = 2, n execută:
  p = p + 1
  s_p = j
  S_a = S_a + a_sp
  S_b = b_j
  dacă S_a > S_b atunci
    p_m = p
    k = p - 1
    cât timp k > 0 execută:
      p_m = ordn(p_m, k)
      k = k - 1
    sfârșit cât timp
    S_a = S_a - a_spm
    pentru i = p_m, p-1 execută:
      s_i = s_i+1
    sfârșit pentru
    p = p - 1
  sfârșit dacă
sfârșit pentru
```

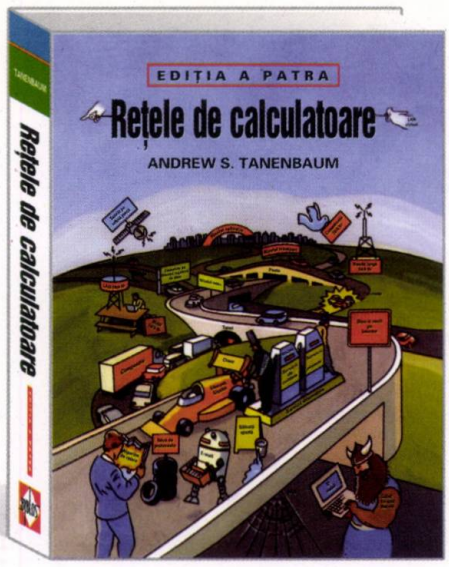
Lucrările care se vor executa fără întârziere sunt în număr de p, iar ordinea optimă se obține prin afișarea elementelor tabloului S.

Algoritmul lui *Johnson* se poate extinde la o problemă de minimizare a timpului total de lucru a n piese pe trei mașini păstrând în continuare aceeași ordine de prelucrare (mai întâi pe mașina M_1, apoi pe M_2 și în final pe M_3); de asemenea, există numeroase alte tipuri de ordonanțare, însă pentru multe dintre ele nu au fost propuși algoritmi eficienți de rezolvare.

Bibliografie

1. Tomescu Ioan, *Combinatorică și teoria grafurilor*, Universitatea București, 1978
2. Tomescu Ioan, *Probleme de combinatorică și teoria grafurilor*, Editura Didactică și Pedagogică București, 1981
3. Rancea Doina, *Algoritmi fundamentali*, Editura Computer Libris Agora, Cluj-Napoca, 1999

Dl. prof. Ilie Vieru este cadru didactic la Colegiul Național "Gh. Vrăncea-nu" din Bacău și poate fi contactat prin e-mail la adresa ivieru@xnet.ro.



Cine are carte, are parte!

... de înțelegerea rețelelor!

Cea mai bună introducere din lume în rețele de calculatoare
- complet actualizată pentru tehnologiile cheie de mâine -

Comand-o acum și dacă ești student primești și o reducere semnificativă!

BYBLOS

Byblos srl, tel: 021-330.92.81, fax: 021-330.92.85, e-mail: dpastrama@byblos.ro
www.byblos.ro
Librăria Eminescu, Universitate, la subsol, tel: 021-312.72.65; 0721-927.499

www.agora.ro
www.pcmagazine.ro
www.eweek.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro

AGORA
pe web

conectează-te
ACUM!

Publicarea acestei reviste este posibilă datorită companiilor de mai jos care sprijină învățământul românesc de informatică

SPONSORI DE PLATINĂ

FORTE COMPUTERS
Flexibility. Made in Romania

IBM®

s&t
IT SOLUTIONS & SERVICES

hp®

net consulting

FUJITSU COMPUTERS SIEMENS

SPONSOR DE AUR

CCT
CABINETUL CĂLIN TATOMIR

Rugăm companiile care doresc să se alăture acestui efort să ne contacteze prin e-mail la adresa office@ginfo.ro sau la numărul de telefon: 0265-166.516